دورة LINQ

Language-Integrated Query

بقلم: محمد سامر أبو سلو

mosamersa@yahoo.com

samerselo2005@hotmail.com

هذا الكتاب يعتبر دورة تعريفية بتقنية Linq نشرح فيها التقنيات الجديدة في فيجول ستوديو 2008 والمتعلقة بهذه التقنية مع شرح لصيغة كتابة الاستعلامات باستخدامها وذلك عبر مزوداتها المختلفة مع العديد من الأمثلة التوضيحية

فهرس الموضوعات

دورة LINQ



مقدمة

تضيف Linq إمكانيات استعلامية بإمكانيات بسيطة وقوية لفيجول بايزيك عندما تتعامل مع العديد من أنواع البيانات المختلفة فبالإضافة إلى إرسال الاستعلام إلى قاعدة بيانات كي تتم معالجته أو العمل مع صيغة مختلفة للاستعلام لكل نوع من أنواع البيانات التي تقوم بالبحث عنها. تقدم Linq الاستعلامات كجزء من لغة فيجول بايزيك مستخدمة صيغة موحدة بغض النظر عن نوع البيانات الذي تستخدمه. وهي تمكنك من الاستعلام عن البيانات من قاعدة بيانات SQL Server أو Xml أو المجموعات والمصفوفات في الذاكرة أو ADO .net Datasets الأمر الذي يجعلها قادرة على الاستعلام من أي قاعدة بيانات يمكن ربطها مع DataSet أو أي مصدر بيانات محلي أو بعيد يدعم Linq حيث يمكنك عمل ذلك كله باستخدام عناصر لغة فيجول بايزيك الشائعة لأن استعلاماتك أصبحت مكتوبة بلغة فيجول بايزيك ونتائج الاستعلام تعاد كأغراض أنواع بيانات قوية داعمة IntelliSense مما يجعل كتابتك للكود أسرع واكتشافك للأخطاء في الاستعلامات عند ترجمة المشروع بدلا من وقت التنفيذ كما أن استعلامات Linq يمكن أن تكون مصدرا لاستعلامات إضافية لمزيد من الدقة في البحث. كما يمكن ربطها مع التحكمات ممكنا المستخدم من استعراض وتعديل نتائج استعلامك بسهولة.

وهذا مثال عن استعلام بسيط يعيد قائمة شركات الزبائن الموجودين في إيطاليا

```
Dim itaCus = From cus In NwDs.Customers _
             Where cus.Country = "Italy" _
             Select cus.ContactName, cus.CompanyName
```

دعنا لا نقلق الآن بخصوص صيغة الاستعلام على كل حال إن كنت متمكنا من كتابة استعلامات سيكول سيرفر لن تجد صعوبة في فهم صيغتها بما أن الصيغة مشابهة مع بعض الاختلافات طبعا والاستعلام السابق يماثل استعلام سيكول سيرفر التالي

SELECT COMPANYNAME FROM CUSTOMERS WHERE COUNTRY='Italy'

كما يمكن أن يكون استعلام Linq أكثر تعقيد فالكود التالي يعيد قائمة بالزبائن ويعيد تجميعهم حسب الموقع

```
Dim customers As List(Of Customer) = GetCustomerList()
Dim customersByCountry = From cust In customers _
                         Order By cust.Country, cust.City _
                         Group By CountryName = cust.Country _
                         Into RegionalCustomers = Group, Count() _
                         Order By CountryName
For Each country In customersByCountry
     Console.WriteLine(country.CountryName & _
          " (" & country.Count & ")" & vbCrLf)
    For Each customer In country.RegionalCustomers
         Console.WriteLine(vbTab & customer.CompanyName & _
              " (" & customer.City & ")")
    Next
Next
```

حيث يمكننا استخدام استعلام Linq في برنامجنا بعدة أشكال فالمثال الأول مثلا يمكننا عرض نتيجته في DataGrid مباشرة

```
Me.DataGridView1.DataSource = itaCus.ToList
```

أو يمكننا إدخاله ضمن حلقة For...Each مثلا واستخدام النتائج ففي المثال التالي نستخدم الاستعلام الوارد بالمثال الأول لإظهار قائمة الشركات في ListBox

```
Me.ListBox1.Items.Clear()
For Each cust In itaCus
       Me.ListBox1.Items.Add(cust.CompanyName)
Next
```

مزودات Linq - LINQ Providers

يقوم مزود Linq بتنظيم استعلامات Linq في فيجول بايزيك بحسب مصدر البيانات الذي تستخدمه فعندما تكتب استعلام Linq يأخذ المزود ذلك الاستعلام ويترجمه إلى أوامر يستطيع مصدر البيانات تنفيذها ويقوم أيضا بتحويل البيانات من الأغراض المصدرية ليشكل نتائج الاستعلام وأخيرا يقوم بتحويل الأغراض إلى بيانات عندما ترسل التحديثات للمصدر. ويضم فيجول بايزيك مزودات Linq التالية:

- Linq to Objects يمكنك هذا المزود من الاستعلام في المجموعات والمصفوفات في الذاكرة إذا كانت غرضك يدعم الواجهة IEnumerable أو الواجهة IEnumerable(T) بحيث يمكنك المزود من الاستعلام عنها ويمكنك تمكين هذا المزود باستيراد المجال System.Linq والذي يكون مستوردا بشكل افتراضي في مشاريع فيجول بايزيك
- Linq to SQL يمكنك هذا المزود من الاستعلام من قواعد بيانات SQL Server والتحديث إليها ويجعل من السهل ربط أغراض التطبيق مع الجداول والأغراض في قواعد البيانات. ويسهل فيجول بايزيك العمل مع Linq to SQL بتقديم Object Relational Designer والذي يمكنك من إنشاء Object Model في التطبيق يرتبط مع الأغراض في قاعدة البيانات ويقدم الـ O/R Designer إمكانية التعامل مع الإجراءات والوظائف المخزنة وإجراءات الغرض DataContext الذي يدير الاتصال مع قاعدة البيانات ويخزن الحالة من أجل تصادم البيانات التفاؤلي
- Linq to Xml يمكنك من الاستعلام من Xml والتعديل عليها بحيث يمكنك تعديل محتويات Xml الموجودة في الذاكرة أو يمكنك تحميل ملف Xml أو حفظه
- Linq to Dataset يمكنك من الاستعلام من ADO .net Datasets والتعديل عليها وإضافة قوة Linq للتطبيقات التي تستخدم Datasets تسهل وتوسع إمكانيات الاستعلام والتجميع والتحديث في الـ Dataset في تطبيقك

بنية استعلامات Linq

يشار عادة إلى استعلام Linq بتعبير الاستعلام وهو يتألف من توليفة من تراكيب الاستعلام التي تحدد مصدر البيانات ومتغيرات التكرار الخاصة بالاستعلام كما يمكنه أن يتضمن تعليمات من أجل الفرز أو التصفية أو التجميع أو الضم أو الحساب ليتم تطبيقها على البيانات المصدرية وصيغتها تكون مشابهة لصيغة الـ SQL ولهذا ستجد أن معظم الصيغة مألوفة.
يبدأ الاستعلام بقسم From الذي يحدد مصدر البيانات والمتغيرات التي تشير إلى كل عنصر من البيانات المصدرية بشكل مستقل وهي تدعى بمتغيرات المجال أو متغيرات التكرار وقسم From مطلوب من أجل الاستعلام إلا في استعلامات التجميع Aggregate حيث يكون قسم From فيها اختياري وبعد تعريف مجال ومصدر الاستعلام في قسم From أو في قسم Aggregate يمكنك تضمين أي تركيب من أقسام الاستعلام. فمثلا الاستعلام التالي يحدد مصدر مجموعة من بيانات الزبائن بالمتغير Customers ومتغير التكرار cust

```
Dim queryResults = From cust In customers _
                   Select cust.CompanyName
```

وهذا المثال يشكل استعلام مقبول بذات نفسه ومع ذلك يصبح الاستعلام أقوى عندما تضيف أقسام استعلام أخرى لتحديد النتائج فمثلا يمكنك إضافة قسم Where لتصفية النتائج إلى قيمة أو أكثر وتكون تعابير الاستعلام عبارة عن سطر واحد من الكود بحيث يمكنك إضافة أقسام استعلام جديدة لنهاية الاستعلام كما يمكنك فصل الاستعلام إلى عدة أسطر لتحسين قراءة كودك باستخدام المحرف _ ويمثل الكود التالي استعلاما يستخدم قسم Where

```
Dim queryResults = From cust In customers _
                   Where cust.Country = "USA"
```

ويمثل قسم select قسم قوي آخر في الاستعلام حيث يمكنك من إعادة الحقول المختارة فقط من مصدر البيانات وتعيد استعلامات Linq مجموعة تعدادية من الأغراض القوية النوع كما يمكنها إعادة أنواع مجهولة أو أنواع معروفة. ويمكن استخدام قسم select للعودة بحقل واحد فقط من مصدر البيانات وعندما تفعل هذا يكون نوع المجموعة المعادة هو نوع بيانات ذلك الحقل. وعندما يعيد قسم Select مجموعة من الحقول من مصدر البيانات تكون المجموعة المعادة من النوع المجهول ويمكنك مطابقة الحقول المعادة من الاستعلام مع حقول من نوع معروف محدد ويظهر الكود التالي تعبير استعلام يعيد مجموعة نوعها مجهول تضم أرقاما مع بيانات من الحقل المحدد من مصدر البيانات

```
Dim queryResults = From cust In customers _
                   Where cust.Country = "USA" _
                   Select cust.CompanyName, cust.Country
```

يمكن استخدام استعلامات Linq لدمج عدة مصادر من البيانات في نتيجة واحدة حيث يمكن عمل هذا باستخدام قسم From واحد أو أكثر أو باستخدام أقسام Join أو Group Join ويظهر الكود التالي تعبير استعلام يضم بيانات Customer و Order ويعيد مجموعة من نوع مجهول تحتوي بيانات من Customer و Order

```
Dim queryResults = From cust In customers, ord In orders _
                   Where cust.CustomerID = ord.CustomerID _
                   Select cust, ord
```

يمكنك استخدام قسم Group Join لبناء استعلامات شجرية تحتوي مجموعة من أغراض Customer وكل غرض Customer يمتلك خاصية تحتوي مجموعة تتضمن جميع أغراض order لذلك الزبون. ويظهر المثال التالي تعبير استعلام يدمج بيانات Customer و Order كنتيجة شجرية ويعيد مجموعة من نوع مجهول ويعيد الاستعلام نوعا يتضمن الخاصية CustomerOrders تحتوي على مجموعة تحتوي على مجموعة من بيانات Order وبيانات Customer وتتضمن أيضا الخاصية OrderTotal تحتوي على مجموع إجمالي الطلبات لذلك الزبون

```
Dim queryResults = From cust In customers _
                   Group Join ord In orders On _
                   cust.CustomerID Equals ord.CustomerID _
                   Into CustomerOrders = Group, _
                        OrderTotal = Sum(ord.Total) _
                   Select cust.CompanyName, cust.CustomerID, _
                          CustomerOrders, OrderTotal
```

معاملات استعلام Visual Basic LINQ Query Operators - Linq

تتضمن الفئات في المجال System.Linq والمجالات التي تدعم Linq طرائق يمكنك استدعاؤها لإنشاء الاستعلامات وتوليها بحسب حاجة التطبيق ويتضمن فيجول بايزيك كلمات مفتاحية لأقسام الاستعلام الشائعة

From Clause

يجب أن يبدأ الاستعلام بقسم From أو Aggregate ويحدد قسم From المجموعة المصدر أو متغير التكرار للاستعلام

```
' Returns the company name for all customers for whom
' State is equal to "WA".
Dim names = From cust In customers _
            Where cust.State = "WA" _
            Select cust.CompanyName
```

Select Clause

اختياري يحدد مجموعة من متغيرات التكرار للاستعلام

```
' Returns the company name and ID value for each
' customer as a collection of a new anonymous type.
Dim customerList = From cust In customers _
                   Select cust.CompanyName, cust.ID
```

و إن لم يكن قسم Select موجودا في الاستعلام فتتألف متغيرات التكرار للاستعلام من تلك المحددة في قسم From أو Aggregate

Where Clause

اختياري ويحدد شرط التصفية للاستعلام

```
' Returns all product names for which the Category of
' the product is "Beverages".
Dim names = From product In products _
            Where product.Category = "Beverages" _
            Select product.Name
```

Order By Clause

اختياري ويحدد ترتيب الفرز للأعمدة في الاستعلام

```
' Returns a list of books sorted by price in
' ascending order.
Dim titlesAscendingPrice = From b In Books _
                           Order By b.Price
```

Join Clause

اختياري ويدمج مجموعتين ضمن مجموعة واحدة

```
Dim Intrst = From i In DsDesposits.InterestRates_
             Join d In DsDesposits.Deposits_
             On i.InterestID Equals d.InterestID_
             Select i.InterestID, i.InterestRate, i.DepositPreiod
```

Group by Clause

اختياري ويقوم بتجميع عناصر نتيجة الاستعلام ويمكن استعماله لتطبيق إجراءات تجميع لكل مجموعة

```
' Returns a list of orders grouped by the order date
' and sorted in ascending order by the order date.
Dim orders = From order In orderList _
             Order By order.OrderDate _
             Group By OrderDate = order.OrderDate _
             Into OrdersByDate = Group
```

Group Join Clause

اختياري ويجمع مجموعتين ضمن مجموعة شجرية واحدة

```
' Returns a combined collection of customers and
' customer orders.
Dim customerList = From cust In customers _
                   Group Join ord In orders On _
                   cust.CustomerID Equals ord.CustomerID _
                   Into CustomerOrders = Group, _
                        OrderTotal = Sum(ord.Total) _
                   Select cust.CompanyName, cust.CustomerID, _
                          CustomerOrders, OrderTotal
```

Aggregate Clause

يجب بدء الاستعلام دوما إما بقسم From أو قسم Aggregate وقسم Aggregate يطبق واحدة أو أكثر من وظائف التجميع على المجموعة فمثلا يمكن استخدام قسم Aggregate لحساب مجموع جميع العناصر المعادة بالاستعلام

```
' Returns the sum of all order totals.
Dim orderTotal = Aggregate order In Orders _
                 Into Sum(order.Total)
```

كما يمكنك استخدام قسم Aggregate لتعديل الاستعلام فمثلا يمكن استخدام قسم Aggregate لإجراء عملية حسابية على مجموعة استعلام

```
' Returns the customer company name and largest
' order total for each customer.
Dim customerMax = From cust In customers _
                  Aggregate order In cust.Orders _
                  Into MaxOrder = Max(order.Total) _
                  Select cust.CompanyName, MaxOrder
```

Let Clause

اختياري ويقوم بحساب قيمة ويضعها في متغير جديد

```
' Returns a list of products with a calculation of
' a ten percent discount.
Dim discountedProducts = From prod In products _
                         Let Discount = prod.UnitPrice * 0.1 _
                         Where Discount >= 50 _
                         Select prod.ProductName, prod.UnitPrice, Discount
```

Distinct Clause

اختياري وهو يضبط القيم المعادة من الاستعلام بحيث لا يجلب قيما مكررة

```
' Returns a list of cities with no duplicate entries.
Dim cities = From item In Customers _
             Select customer.City_
             Distinct
```

Skip Clause

اختياري يتجاوز عددا معينا من العناصر في المجموعة ويعيد الباقي

```
' Returns a list of customers. The first 10 customers
' are ignored and the remaining customers are
' returned.
Dim customerList = From cust In customers _
                   Skip 10
```

Skip While Clause

اختياري يتجاوز عناصر المجموعة طالما قيمة الشرط True ثم يعيد باقي العناصر

```
' Returns a list of customers. The query ignores all
' customers until the first customer for whom
' IsSubscriber returns false. That customer and all
' remaining customers are returned.
Dim customerList = From cust In customers _
                   Skip While IsSubscriber(cust)
```

Take Clause

اختياري ويعيد عددا من العناصر المتجاورة في بداية المجموعة

```
' Returns the first 10 customers.
Dim customerList = From cust In customers _
                   Take 10
```

Take While Clause

اختياري يقوم بتضمين عناصر المجموعة طالما قيمة الشرط True ويتجاهل بقية العناصر

```
' Returns a list of customers. The query returns
' customers until the first customer for whom
' HasOrders returns false. That customer and all
' remaining customers are ignored.
Dim customersWithOrders = From cust In customers _
                          Order By cust.Orders.Count Descending _
                          Take While HasOrders(cust)
```

كما يمكنك استخدام خصائص إضافية لاستعلام Linq باستدعاء عناصر المجموعات والأنواع المستعلم عنها التي يوفرها Linq حيث يمكنك استخدام هذه الإمكانيات الإضافية باستدعاء معامل استعلام على نتيجة الاستعلام فمثلا الكود التالي يستخدم الطريقة Union لدمج ناتج استعلامين في نتيجة استعلام واحدة ويستخدم الطريقة ToList(TSource) لإعادة ناتج الاستعلام كقائمة

```
Public Function GetAllCustomers() As List(Of Customer)
    Dim customers1 = From cust In domesticCustomers
    Dim customers2 = From cust In internationalCustomers

    Dim customerList = customers1.Union(customers2)
    Return customerList.ToList()
End Function
```

مواضيع متعلقة بتقنية Linq لابد من الإطلاع عليها

الاستدلال المحلي على النوع Local Type Inference

يستخدم المترجم في فيجول بايزيك 2008 الاستدلال على النوع Type Inference لتحديد نوع المتغيرات المحلية التي تم التصريح عنها بدون استخدام فقرة As في تعبير التصريح حيث يستدل المترجم على نوع المتغير من نوع التعبير الذي يضبط قيمة ذلك المتغير مما يوفر إمكانية تعريف المغيرات بدون تحديد نوعها كما في المثال التالي

```
Public Sub inferenceExample()

    ' Using explicit typing .
    Dim num1 As Integer = 3

    ' Using local type inference.
    Dim num2 = 3

End Sub
```

ولا يمكن استخدام الاستدلال على النوع عند تعريف الحقول في الفئة Class Fields فإن كان num2 في المثال السابق حقلا في فئة بدلا عن كونه متغيرا محليا فسوف يولد التصريح خطأ إذا كان Option Strict On وسوف يصنف num2 على أنه غرض Object إن كان Option Strict Off وبشكل مشابه فنوع المتغيرات الساكنة Static Variables لا يمكن الاستدلال عليها إن كان Option Strict On وإن كان Option Strict Off فنوع المتغير الساكن سيكون غرض Object فإن لم تكن تريد من المتغير num2 في المثال السابق أن يكون من النوع Integer فيمكنك تحديد نوعا آخر عند التصريح عنه

Dim num3 As Object = 3 or Dim num4 As Double = 3

والكود الذي يستخدم استدلال النوع يشابه الكود الذي يعتمد على الربط المتأخر Late Binding الذي سيكون نوعه معروفا فقط في زمن التشغيل. ومعرفة النوع بشكل مبكر يمكن المترجم من تحديد المشاكل قبل التنفيذ وحجز الذاكرة بدقة وإجراء عمليات التحسين الأخرى بالإضافة إلى تمكين بيئة التطوير من تزويد المبرمج بـ IntelliSense والمساعدة حول أعضاء ذلك الغرض بالإضافة إلى تفضيله لاعتبارات خاصة بالأداء بسبب أن جميع البيانات التي تخزن باستخدام الربط المتأخر يجب تغليفها وكأنها من النوع Object والوصول إلى الأعضاء في زمن التشغيل سيكون أبطأ.

يحدث الاستدلال على النوع عندما يتم التصريح عن المتغير بدون استخدام فقرة As في تعبير التصريح وضبط قيمة لذلك المتغير فيستخدم المترجم نوع تلك القيمة كنوع للمتغير فمثلا سطور الكود التالية تعرف متغيرا من النوع String

```
' Using explicit typing.
Dim name1 As String = "Springfield"

' Using local type inference.
Dim name2 = "Springfield"
```

ويستعرض الكود التالي طريقتان متكافئتان لإنشاء مصفوفة من النوع Integer

```
' Using explicit typing.
Dim someNumbers1() As Integer = New Integer() {4, 18, 11, 9, 8, 0, 5}

' Using local type inference.
Dim someNumbers2 = New Integer() {4, 18, 11, 9, 8, 0, 5}
```

كما يمكنك استخدام الاستدلال على النوع لتحديد نوع متغير التحكم لحلقة تكرارية ففي الكود التالي سيتعرف المترجم على num بأنه من النوع Integer لأن someNumbers2 عبارة عن مصفوفة Integer

```
Dim total = 0
For Each number In someNumbers2
    total += number
Next
```

ويستخدم الاستدلال على النوع في العبارة Using أيضا لتحديد نوع اسم المصدر كما هو واضح في المثال التالي

```
Using proc = New System.Diagnostics.Process
    ' Insert code to work with the resource.
End Using
```

ويستدل على نوع المتغير من القيمة المعادة من الإجراء أيضا كما هو ظاهر في الكود التالي حيث يكون pList1 و pList2 عبارة عن Lists of Processes

```
' Using explicit typing.
Dim pList1() As Process = Process.GetProcesses()

' Using local type inference.
Dim pList2 = Process.GetProcesses()
```

وقد قدم فيجول بايزيك 2008 خيارا جديدا هو Option Infer يمكنك من تحديد إذا كان الاستدلال المحلي على النوع مسموحا أم لا في ملف معين. فلتمكين أو تعطيل خيار الاستدلال على النوع اكتب التعبير المناسب من السطرين التاليين في بداية الملف

```
Option Infer On
Option Infer Off
```

وإن لم تقم بتحديد قيمة للخيار Option Infer في الكود فالمترجم سيستخدم الخيار الافتراضي Option Infer On من أجل المشاريع التي تم إنشاؤها في Visual Basic 2008 والخيار Option Infer Off من أجل المشاريع التي تمت ترقيتها من إصدارات سابقة. وإن تضاربت قيمة الخيار Option Infer في الملف مع القيمة المضبوطة في خيارات بيئة التطوير أو في سطر الأوامر فسوف يتم استخدام القيمة الموجودة في الملف.

ويستخدم الاستدلال على النوع فقط في المتغيرات الغير ساكنة Non-Static ولا يمكن استخدامها في تعريف حقول الفئة Class Fields أو الخصائص Properties أو الإجراءات Functions

الأنواع المجهولة Anonymous Types

يقدم فيجول ستوديو 2008 الأنواع المجهولة anonymous types والتي تمكنك من إنشاء الأغراض Objects بدون كتابة تعريف فئة Class definition من أجل نوع البيانات وعوضا عن ذلك يولد المترجم الفئة من أجلك ولن يكون للفئة اسما قابلا للاستخدام حيث تكون هذه الفئات موروثة مباشرة من Object وتمتلك الخصائص التي تحددها عند تعريف الغرض Object وبما أن نوع البيانات لم يتم تحديده يتم الإشارة إليه على أنه نوع مجهول anonymous type. حيث يصرح المثال التالي عن المتغير product كمتغير من النوع anonymous type ممتلكا الخاصيتين Name و Price

```
' Variable product is an instance of a simple anonymous type.
Dim product = New With {Key .Name = "paperclips", .Price = 1.29}
```

حيث يستخدم تعبير الاستعلام التالي الأنواع المجهولة لدمج أعمدة البيانات المحددة بواسطة الاستعلام وبما أنه لا يمكنك تحديد نوع النتيجة مقدما بسبب عدم إمكانية التنبؤ بالأعمدة التي يمكن أن يختارها استعلام معين فتمكنك الأنواع المجهولة من كتابة استعلام يختار عدد من الأعمدة بأي ترتيب نريده فيقوم المترجم بإنشاء نوع البيانات المماثل لتلك الخصائص المحددة بذلك الترتيب المعين. وفي المثال التالي يكون Products عبارة عن قائمة من أغراض Product وكل منها يمتلك خصائص عديدة بحيث يحمل المتغير namePriceQuery تعريف الاستعلام الذي يعيد عند تنفيذه مجموعة من الأنواع المجهولة التي تمتلك الخاصيتين Name و Price

```
Dim namePriceQuery = From prod In products_
              Select prod.Name, prod.Price
```

والمتغير nameQuantityQuery يحمل تعريف الاستعلام الذي يعيد عند تنفيذه مجموعة من الأنواع المجهولة التي تمتلك خاصيتين Name و OnHand

```
Dim nameQuantityQuery = From prod In products_
                Select prod.Name, prod.OnHand
```

تعريف نوع مجهول Declaring an Anonymous Type

تعريف متغير من نوع مجهول يستخدم قائمة بناء لتحديد خصائص ذلك النوع بحيث يمكنك تحديد هذه الخصائص فقط عند الإعلان عن النوع المجهول ولا يمكن استخدام بقية عناصر الفئات مثل الطرائق والأحداث في الأنواع المجهولة ففي المثال التالي يكون Product1 من نوع مجهول يمتلك خاصيتين Name و Price

```
'Variable product1 is an instance of a simple anonymous type.
Dim product1 = New With {.Name = "paperclips", .Price = 1.29}

- 'or-

'product2 is an instance of an anonymous type with key properties.
Dim product2 = New With {Key .Name = "paperclips", Key .Price = 1.29}
```

فإن قمت بتحديد الخصائص كخصائص مفتاحية key properties أصبح بإمكانك استخدامها لمقارنة نوعين مجهولين هل هما متساويين أم لا ومع ذلك فقيم الخصائص المفتاحية لا يمكن تغييرها فهي للقراءة فقط. مع ملاحظة أن التصريح عن نوع مجهول يماثل التصريح عن نوع مسمى باستخدام باني الغرض object initializer

```
'Variable product3 is an instance of a class named Product.
Dim product3 = New Product With {.Name = "paperclips", .Price = 1.29}
```

الخصائص المفتاحية Key Properties

تختلف الخصائص المفتاحية عن العادية بعدة أمور:

- تستخدم الخصائص المفتاحية فقط لمقارنة المساواة بين نوعين مجهولين
- لا يمكن تغيير قيم الخصائص المفتاحية فهي دائما للقراءة فقط
- فقط الخصائص المفتاحية يتم تضمينها ضمن الـ Hash Code الذي يولده المترجم من أجل الأنواع المجهولة

المساواة Equality

تكون متغيرات الأنواع المجهولة متساوية عندما تكون متغيرات لنفس النوع المجهول ويقوم المعالج بمعاملة متغيرين كمتغيرين من نفس النوع إذا توفرت فيهما الشروط التالية

- تم التصريح عنهما في نفس المجمع
- تمتلك خصائصهما نفس الاسم والنوع وتم التصريح عنها بنفس الترتيب وتكون مقارنة الأسماء غير حساسة لحالة الأحرف
- نفس الخصائص فيها محددة كخصائص أساسية
- يمتلك كل نوع خاصية أساسية واحدة على الأقل

والتصريح عن نوع مجهول الذي لا يمتلك أي خاصية مفتاحية يكون مساويا لنفسه فقط

```
'prod1 and prod2 have no key values.
Dim prod1 = New With {.Name = "paperclips", .Price = 1.29}
Dim prod2 = New With {.Name = "paperclips", .Price = 1.29}

'The following line displays False, because prod1 and prod2 have no
'key properties.
Console.WriteLine(prod1.Equals(prod2))

'The following statement displays True because prod1 is equal to itself.
Console.WriteLine(prod1.Equals(prod1))
```

وتكون قيمة متغيرين لنفس النوع المجهول متساويين إذا كانت قيمة خصائصهما المفتاحية متساوية كما في المثال التالي الذي يوضح كيفية فحص هذه المساواة

```
Dim prod3 = New With {Key .Name = "paperclips", Key .Price = 1.29}
Dim prod4 = New With {Key .Name = "paperclips", Key .Price = 1.29}
'The following line displays True, because prod3 and prod4 are
'instances of the same anonymous type, and the values of their
'key properties are equal.
Console.WriteLine(prod3.Equals(prod4))

Dim prod5 = New With {Key .Name = "paperclips", Key .Price = 1.29}
Dim prod6 = New With {Key .Name = "paperclips", Key .Price = 1.29, OnHand = 423}
'The following line displays False, because prod5 and prod6 do not
'have the same properties.
Console.WriteLine(prod5.Equals(prod6))

Dim prod7 = New With {Key .Name = "paperclips", Key .Price = 1.29, OnHand = 24}
Dim prod8 = New With {Key .Name = "paperclips", Key .Price = 1.29, OnHand = 423}
'The following line displays True, because prod7 and prod8 are
'instances of the same anonymous type, and the values of their
'key properties are equal. The equality check does not compare the
'values of the non-key field.
Console.WriteLine(prod7.Equals(prod8))
```

القيم القابلة للقراءة فقط

لا يمكن تغيير قيم الخصائص المفتاحية فمثلا في prod8 في المثال السابق الحقول Name و Price قابلة للقراءة فقط في حين أن الحقل OnHamd يمكن تغيير قيمته

```
'The following statement will not compile, because Name is a key
'property and its value cannot be changed.
'prod8.Name = "clamps"
```

```
'OnHand is not a Key property. Its value can be changed.
prod8.OnHand = 22
```

الأنواع المجهولة من تعابير الاستعلام Anonymous Types from Query Expressions

تعابير الاستعلام لا تتطلب دوما إنشاء أنواع مجهولة فعند الإمكان يمكنها استخدام نوع موجود ليحمل بيانات العمود وهذا يحدث عندما يعيد الاستعلام إما سجلات كاملة من مصدر البيانات أو حقل واحد من كل سجل ففي المثال التالي يكون Customers عبارة عن مجموعة فئات Customer والفئة تمتلك العديد من الخصائص بحيث يمكنك تضمين واحدة أو أكثر من هذه الخصائص في نتائج الاستعلام وبأي ترتيب تريده ففي المثالين الأوليين لا يوجد حاجة لأي نوع مجهول لأن الاستعلام يجلب عناصر من أنواع معروفة فـ Custs1 يكون من النوع string لأن cust.Name من النوع String و Custs2 هو مجموعة من الأغراض Customers لأن كل عنصر في Customers هو غرض Customer وكامل العنصر تم جلبه بواسطة الاستعلام

```
Dim custs1 = From cust In customers_
          Select cust.Name

Dim custs2 = From cust In customers_
          Select cust
```

ومع ذلك فالأنواع المسماة لا تكون دائما متوفرة حيث يمكنك الاستعلام عن Names و Addresses من أجل هدف معين و ID و Numbers و Location من أجل هدف آخر فهنا تمكنك الأنواع المجهولة من اختيار أية تركيبة من الخصائص وبأي ترتيب بدون أن تضطر في البداية للتصريح عن نوع مسمى جديد ليحمل النتيجة وبدلا عن ذلك يقوم المترجم بإنشاء نوع مجهول لكل تركيبة من الخصائص فمثلا الاستعلام التالي يحدد فقط Name و ID من كل غرض Customer في customers ومن أجل ذلك يقوم بإنشاء نوع مجهول من تلك الخصائص

```
Dim custs3 = From cust In customers_
          Select cust.Name, cust.ID
```

وكل من الاسم والنوع العائدين لخصائص النوع المجهول يتم أخذها من بارامترات الاستعلام cust.Name و Cust.Id وتكون خصائص النوع المجهول التي ينشئها الاستعلام خصائص مفتاحية دوما وعند تنفيذ cust3 في حلقة For...Each التالية تكون النتيجة هي مجموعة أنواع مجهولة تمتلك خاصيتين مفتاحيتين Name و ID

```
For Each selectedCust In custs3
   Console.WriteLine(selectedCust.ID & ": " & selectedCust.Name(
Next
```

تحديد متى نستخدم الأنواع المجهولة

قبل أن تقوم بالتصريح عن غرض بأنه من نوع مجهول يجب عليك التفكير فيما إذا كان هذا الخيار هو الأفضل فمثلا إن كنت تريد إنشاء غرض مؤقت ليحتوي بعض حقول المعلومات ولست بحاجة إلى بقية الحقول والطرائق التي تحتويها الفئة الكاملة يكون عندها النوع المجهول حلا جيدا وتكون الأنواع المجهولة ملائمة عندما تريد انتقاء مجموعة مختلفة من الخصائص عند كل تصريح أو إن كنت تريد تغيير ترتيب هذه الخصائص وإن كان مشروعك يحتوي على عدة أغراض تحمل نفس الخصائص بترتيب ثابت يمكنك عندها التصريح عنهم بسهولة باستخدام الأنواع المسماة باستخدام باني فئة فعندها باستخدام باني ملائم يمكن تعريف عدة متغيرات من الفئة Product ويكون ذلك أسهل من استخدام عدة متغيرات مجهولة النوع

```
'Declaring instances of a named type.
Dim firstProd1 As New Product("paperclips", 1.29)
Dim secondProd1 As New Product("desklamp", 28.99)
Dim thirdProd1 As New Product("stapler", 5.09)
'Declaring instances of an anonymous type.
Dim firstProd2 = New With {Key .Name = "paperclips", Key .Price = 1.29}
Dim secondProd2 = New With {Key .Name = "desklamp", Key .Price = 28.99}
Dim thirdProd2 = New With {Key .Name = "stapler", Key .Price = 5.09}
```

وتكمن فائدة أخرى للأنواع المجهولة في أن المترجم يمكنه التقاط الأخطاء الطباعية في أسماء الخصائص ففي المثال السابق يفترض بالأنواع firstProd2 و secondProd2 و thirdProd2 أن تكون متغيرات لنفس النوع المجهول ومع ذلك قمت عن طريق الخطأ بالتصريح عن thirdProd2 بأحد الطرائق اللاحقة وهو نوع مختلف عن firstProd2 و secondProd2

```
'Dim thirdProd2 = New With {Key .Nmae = "stapler", Key .Price = 5.09}
'Dim thirdProd2 = New With {Key .Name = "stapler", Key .Price = "5.09"}
```

```
'Dim thirdProd2 = New With {Key .Name = "stapler", .Price = 5.09}
```

والأمر الأهم هو أنه هناك حدود لاستخدام الأنواع المجهولة لا تنطبق على الأنواع المعروفة فمع أن firstProd2 و secondProd2 و thirdProd2 هي متغيرات لنفس النوع المجهول فالمتغير المجهول المشترك غير متوفر ولا يمكن توقع ظهوره كنوع معروف في الكود فمثلا يمكن استخدام النوع المجهول لتحديد توقيع الطريقة للتصريح عن حقل متغير فيكون بالنتيجة النوع المجهول غير ملائم لتبادل البيانات عبر الطرائق

Lambda Expressions

الـ Lambda Expression هو وظيفة Function بدون اسم تحتسب وتعيد قيمة وحيدة كما يمكن استخدامها في التعابير التي تطلب إجراءات مفوضة Delegate والمثال التالي عن هذه التعابير يأخذ قيمة ويعيد الناتج بعد إضافة واحد لها

```
Function (num As Integer) num + 1
```

كما يمكنك إسناد هذه الوظيفة لمتغير وتمرير القيمة له

```
Dim add1 = Function(num As Integer) num + 1
Console.WriteLine(add1(5))
```

كما يمكنك تعريف وتنفيذ الوظيفة بنفس الوقت

```
Console.WriteLine((Function(num As Integer) num + 1)(5))
```

كما يمكن أن تستخدم Lambda Expressions كقيمة معادة عند استدعاء وظيفة أو تمريرها كوسيط لإجراء مفوض ففي المثال التالي تستخدم Lambda Expressions بوليانية كوسائط للإجراء testResult حيث تطبق الطريقة فحص بولياني لوسيط من النوع Integer ويظهر القيمة Success إذا كانت قيمة Lambda Expression هي True أو Failure إن كانت قيمته False

```
Module Module2

    Sub Main()
        ' The following line will print Success, because 4 is even.
        testResult(4, Function(num) num Mod 2 = 0)
        ' The following line will print Failure, because 5 is not > 10.
        testResult(5, Function(num) num > 10)
    End Sub

    ' Sub testResult takes two arguments, an integer value and a
    ' Boolean function.
    ' If the function returns True for the integer argument, Success
    ' is displayed.
    ' If the function returns False for the integer argument, Failure
    ' is displayed.
    Sub testResult(ByVal value As Integer, ByVal fun As Func(Of Integer,
Boolean))
        If fun(value) Then
            Console.WriteLine("Success")
        Else
            Console.WriteLine("Failure")
        End If
    End Sub

End Module
```

تكون تعابير Lambda Expressions هي الأساس لكثير من معاملات الاستعلام Linq حيث يقوم المترجم Compiler بإنشاء تعابير Lambda Expressions للقيام بالعمليات الحسابية للطرائق الخاصة بالاستعلام مثل Where و Select و Order By و Take While فعلى سبيل المثال انظر الاستعلام التالي

```
Dim londonCusts = From cust In db.Customers
                  Where cust.City = "London"
                  Select cust
```

حيث ستتم ترجمته إلى الكود التالي

```
Dim londonCusts = db.Customers _
    .Where(Function(cust) cust.City = "London") _
    .Select(Function(cust) cust)
```

وتكون صيغتها على الشكل

- هذه التعابير لا تملك اسما
- لا يمكن استخدام المعدلات معها مثل Overloads أو Overrides
- لا تستخدم قسم AS لتحديد نوع القيمة المعادة وبدلا عن ذلك يكون نوع القيمة المعادة هو نوع القيمة التي يشكلها جسم الإجراء فإن كان جسم الإجراء مثلا Cust.City = "London" فتكون القيمة المعادة بوليانية
- جسم الإجراء يجب أن يكون تعبير وليس تصريح ويمكن أن يحتوي على استدعاء لوظيفة Function ولكنه لا يمكن أن يستدعي إجراء Sub
- لا يوجد تعبير Return وتكون القيمة المعادة هي قيمة ذلك التعبير الذي يشكل جسم الوظيفة
- لا يوجد تعبير End
- يجب أن تكون جميع الوسائط محددة النوع أو تكون جميعها بأنواع بالإشارة
- غير مسموح بالوسائط الاختيارية
- الوسائط Generic غير مسموح بها

ونتيجة لهذه القواعد سنرى أن أي تعبير Lambda Expression سيكون بسيطا وغير معقد

تشترك Lambda Expression مع الوظائف Methods بأنها محددة ولها جميع حقوق الوصول كأي كود مكتوب في الطريقة التي تحتويها وهذا يتضمن الوصول إلى متغيرات الأعضاء والوظائف وجميع المغيرات الموجودة في الوظيفة التي تحتوي التعبير Lambda Expression ففي المثال التالي المتغير target هو محلي بالنسبة لـ makeTheGame والطريقة التي تم تحديد التعبير Lambda Expression فيها هي playTheGame لاحظ أن القيمة المعادة من التعبير Lambda Expression يتم تعيينها لـ takeAGuess في Main مازالت تستطيع الوصول للمتغير المحلي target

```
Module Module1

    Sub Main()
       ' Variable takeAGuess is a Boolean function. It stores the target
       ' number that is set in makeTheGame.
       Dim takeAGuess As gameDelegate = makeTheGame()

       ' Set up the loop to play the game.
       Dim guess As Integer
       Dim gameOver = False
       While Not gameOver
         guess = CInt(InputBox("Enter a number between 1 and 10 (0 to quit)", "Guessing Game", "0"))
            ' A guess of 0 means you want to give up.
            If guess = 0 Then
                gameOver = True
            Else
                ' Tests your guess and announces whether you are correct. Method takeAGuess
                ' is called multiple times with different guesses. The target value is not
                ' accessible from Main and is not passed in.
                gameOver = takeAGuess(guess)
                Console.WriteLine("Guess of " & guess & " is " & gameOver)
            End If
        End While

    End Sub

    Delegate Function gameDelegate(ByVal aGuess As Integer) As Boolean

    Public Function makeTheGame() As gameDelegate

        ' Generate the target number, between 1 and 10. Notice that
        ' target is a local variable. After you return from makeTheGame,
        ' it is not directly accessible.
        Randomize()
        Dim target As Integer = CInt(Int(10 * Rnd() + 1))

        ' Print the answer if you want to be sure the game is not cheating
        ' by changing the target at each guess.
        Console.WriteLine("(Peeking at the answer) The target is " & target)

        ' The game is returned as a lambda expression. The lambda expression
        ' carries with it the environment in which it was created. This
        ' environment includes the target number. Note that only the current
        ' guess is a parameter to the returned lambda expression, not the target.

        ' Does the guess equal the target?
        Dim playTheGame = Function(guess As Integer) guess = target
```

```
        Return playTheGame
    End Function
End Module
```

ويستعرض المثال التالي مجالا عريضا من حقوق الوصول المعششة في Lambda Expression فعندما يتم تنفيذ التعبير Lambda Expression من Main كـ aDel يستخدم العناصر التالية (حقل في الفئة aField – خاصية في الفئة aProp – وسيط للإجرائية functionWithNestedLambda هو level1 – متغير محلي لـ functionWithNestedLambda هو localVar – وسيط للتعبير Lambda Expression المعشش هو level2)

```
Module Module3

    Sub Main()
        ' Create an instance of the class, with 1 as the value of
        ' the property.
        Dim lambdaScopeDemoInstance = New LambdaScopeDemoClass _
            With {.Prop = 1}

        ' Variable aDel will be bound to the nested lambda expression
        ' returned by the call to functionWithNestedLambda.
        ' The value 2 is sent in for parameter level1.
        Dim aDel As aDelegate = _
            lambdaScopeDemoInstance.functionWithNestedLambda(2)

        ' Now the returned lambda expression is called, with 4 as the
        ' value of parameter level3.
        Console.WriteLine("First value returned by aDel:    " & aDel(4))

        ' Change a few values to verify that the lambda expression has
        ' access to the variables, not just their original values.
        lambdaScopeDemoInstance.aField = 20
        lambdaScopeDemoInstance.Prop = 30
        Console.WriteLine("Second value returned by aDel: " & aDel(40))
    End Sub

    Delegate Function aDelegate(ByVal delParameter As Integer) _
        As Integer

    Public Class LambdaScopeDemoClass
        Public aField As Integer = 6
        Dim aProp As Integer

        Property Prop() As Integer
            Get
                Return aProp
            End Get
            Set(ByVal value As Integer)
                aProp = value
            End Set
        End Property

        Public Function functionWithNestedLambda _
           (ByVal level1 As Integer) As aDelegate
            Dim localVar As Integer = 5

            ' When the nested lambda expression is executed the first
            ' time, as aDel from Main, the variables have these values:
            ' level1 = 2
            ' level2 = 3, after aLambda is called in the Return statement
            ' level3 = 4, after aDel is called in Main
            ' locarVar = 5
            ' aField = 6
            ' aProp = 1
```

```
            ' The second time it is executed, two values have changed:
            ' aField = 20
            ' aProp = 30
            ' level3 = 40
            Dim aLambda = Function(level2 As Integer) _
                              Function(level3 As Integer) _
                                  level1 + level2 + level3 + localVar _
                                  + aField + aProp

            ' The function returns the nested lambda, with 3 as the
            ' value of parameter level2.
            Return aLambda(3)
        End Function

    End Class
End Module
```

كما يمكن تحويل Lambda Expressions لتتوافق مع الإجراءات المفوضة فعندما تعين Lambda Expression لإجراء مفوض Delegate يمكنك تحديد أسماء الوسائط ولكن مع إغفال أنواع البيانات الخاصة بها تاركا مهمة تحديدها للإجراء المفوض ففي المثال التالي يتم تعيين Lambda Expression لمتغير اسمه del من النوع ExampleDel الذي هو عبارة عن إجراء مفوض يأخذ وسيطتين integer و string لاحظ أن أنواع المتغيرات في Lambda Expression لم يتم تحديدها ومع ذلك فـ del يتطلب وسيطا من النوع integer ووسيطا آخر من النوع string كما تم تحديده عند تعريف ExampleDel

```
' Definition of function delegate ExampleDel.
Delegate Function ExampleDel(ByVal arg1 As Integer, _
                             ByVal arg2 As String) As Integer

' Declaration of del as an instance of ExampleDel, with no data
' type specified for the parameters, m and s.
Dim del As ExampleDel = Function(m, s) m

' Valid call to del, sending in an integer and a string.
Console.WriteLine(del(7, "up"))

' Neither of these calls is valid. Function del requires an integer
' argument and a string argument.
'Console.WriteLine(del(7, 3))
'Console.WriteLine(del("abc"))
```

في المثال التالي يتم تحديد Lambda Expression ليعيد القيمة True إذا كان الوسيط يمتلك قسمة أو False إذا كان القيمة Nothing

```
Dim notNothing = Function(num? As Integer) _
             num IsNot Nothing
Dim arg As Integer = 14
Console.WriteLine("Does the argument have an assigned value?")
Console.WriteLine(notNothing(arg))
```

والمثال التالي يحدد Lambda Expression يعيد Index العنصر الأخير في مصفوفة

```
Dim numbers() As Integer = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
Dim lastIndex = Function(intArray() As Integer) _
                    intArray.Length - 1
For i = 0 To lastIndex(numbers)
    numbers(i) = numbers(i) + 1
Next
```

تعابير لمدا في العمق Lambda Expressions

تعتبر تعابير لمدا من الإضافات المفيدة في فيجول بايزيك 2008 حيث يمكنك إعادتها كقيمة من وظيفة أو تمريرها كمحدد لوظيفة أخرى حيث تمت إضافتها للغة البايزيك كدعم للغة الاستعلامات المضمنة Linq التي تضيف إمكانيات استعلامية قوية لبرمجة البيانات في فيجول بايزيك وعندما تبدأ باستخدام تعابير لمدا سترى القوة والمرونة الكامنة فيها

ما هي تعابير لمدا

يشكل الكود التالي مثالا عن تعريف تعبير لمدا أساسي فهو يعرف تعبير لمدا يأخذ Integer ويعيد Integer بحيث يأخذ قيمة الدخل ويعيدها مضروبة بـ 2

```
Dim doubleIt as Func(Of Integer, integer) = _
    Function(x As Integer) x * 2
```

والنوع Func من الأنواع الجديدة في فيجول بايزيك 2008 وهو في الأساس إجراء مفوض Delegate يعيد نوعا يحدده المحدد الأخير ويمكنك من تمرير أربعة محددات تسبق ذلك المحدد والنوع المفوض Func معرف في المجمع System.Core.dll الأمر الذي يمكنك من الاستفادة منه فوريا وذلك لأن المجمع المذكور يتم استيراده تلقائيا عندما تنشئ تطبيقا جديدا ويمثل الكود التالي تحميلات Overloads مختلفة لـ Func

```
Dim f0 As Func(Of Boolean)
Dim f1 As Func(Of Integer, Boolean)
Dim f4 As Func(Of Integer, Integer, Integer, Integer, Boolean)
```

ففي المثال السابق f0 هو مفوض يعيد قيمة Boolean و f1 مفوض يأخذ Integer ويعيد Boolean و f4 مفوض يأخذ أربعة محددات من النوع Integer ويعيد قيمة من النوع Boolean وتكمن النقطة الأساسية في التعبير لمدا هو أنه مفوض قابل للاستدعاء تماما كالمفوضات في فيجول بايزيك 2005 فمن ناحية المساواة اليمنى في قطعة الكود الأولى يمكنك رؤية الصيغة الجديدة للتعابير لمدا فهي تبدأ بالكلمة المحجوزة Function متبوعة بقائمة من المحددات وتعبير وحيد ففي المثال السابق يأخذ تعبير لمدا محددا واحدا من النوع Integer ونلاحظ عدم وجود تعبير Return وذلك لأن المترجم يعرف النوع المؤسس عليه التعبير وبهذا يقوم بتمرير عبارة Return تلقائيا وفي هذه الحالة بما أن x هو من النوع Integer ونتيجة المعادلة هي Integer لهذا فنتيجة تعبير لمدا هي Integer أيضا ويمكن السحر في تعابير لمدا في أنه يمكن استخدامها كمفوض بسيط كما نرى في المثال

```
Dim doubleIt As Func(Of Integer, Integer) = _
    Function(x As Integer) x * 2
Dim z = doubleIt(20)
```

فإن نفذت الكود السابق سترى أن القيمة المخزنة في z هي 40 وأنت بهذا قمت بإنشاء تعبير لمدا يضاعف قيمة أي Integer يمرر له.

دعنا الآن نتفحص مثال معقد أكثر باستخدام تعابير لمدا

```
Dim mult As Func(Of Integer, Func(Of Integer, Integer)) = _
    Function(x As Integer) Function(y As Integer) x * y
```

ويعتبر mult تعبير لمدا معقد قليلا فهو يأخذ كدخل له محدد من النوع Integer ويعيد تعبير لمدا كقيمة له والذي أخذ بدوره قيمة Integer ويعيد قيمة Integer كما يمكننا إعادة تقسيم التعبير السابق على أسطر من أجل توضيح الكود

```
Dim mult As Func(Of Integer, Func(Of Integer, Integer)) = _
    Function(x As Integer)  _
        Function(y As Integer) x * y
```

فتعبير لمدا الخارجي يحتوي تعبير لمدا آخر الذي يستخدم من قبل المترجم كقيمة معادة ويكون التوقيع الخاص بتعبير لمدا الداخلي مماثلا لتوقيع المفوض Func(Of Integer, Integer) في القيمة المعادة من تعبير لمدا الخارجي حيث يقوم المترجم بترجمة التعبير بأكمله دون مشاكل ويمكننا رؤية تعبير لمدا هذا كما يلي

```
Dim mult_10 = mult(10)
Dim r = mult_10(4)
```

فالسطر الأول يحدد mult_10 كـ mult(10) وبما أن Mult(10) يعيد تعبير لمدا يأخذ محدد ويضربه بـ 10 والنوع المعاد من mult_10 هو Func(Of Integer, Integer) والسطر الثاني يستدعي mult_10 ممررا له القيمة 4 بهذا ستكون القيمة المخزنة في r هي 40 ويكون نوع r هو Integer ويعتبر mult مصنع لتعابير لمدا فهو يعيد تعبير لمدا مخصص بالمحدد الأول وستلاحظ أن تعبير لمدا الداخلي يستخدم محدد تعبير لمدا الخارجي ولكن فترة حياة تعبير لمدا الداخلي تتجاوز فترة حياة تعبير لمدا الخارجي

تعابير لمدا كاستدعاءات

بما أن تعابير لمدا هي ببساطة مفوضات لذا يمكنك استخدامها في أي مكان يمكن استخدام المفوض فيه. لاحظ الإجراء التالي الذي يأخذ مفوض كمحدد له ويستدعي مفوض من أجل كل عنصر في القائمة

```
Delegate Function ShouldProcess(Of T) (element As T) As Boolean

Sub ProcessList(Of T) ( _
            Elements As List(Of T), shouldProcess As ShouldProcess(Of T))

    For Each elem in elements
        If shouldProcess(elem) Then
            ' Do some processing here
        End If
    Next
End Sub
```

ويكون المثال التالي تطبيقا قياسيا على المفوضات فالطريقة ProcessList ستمر على كل عنصر من القائمة وتتحقق فيما إذا كان عليها معالجة العنصر ثم تقوم ببعض المعالجة وحتى تتمكن من استخدام هذا في فيجول بايزيك 2005 عليك إنشاء وظيفة تمتلك نفس توقيع المفوض ثم تمرر عنوان تلك الوظيفة إلى الإجراء ProcessList

```
Class Person
    Public age As Integer
End Class

Function _PrivateShouldProcess(person As Person) As Boolean
    Return person.age > 50
End Function

Sub DoIt()
    Dim list As New List(Of Person)
    'Obtain list of Person from a database, for example
    ProcessList(list, AddressOf _PrivateShouldProcess)
End Sub
```

وهذا يسبب بعض الإزعاج فغالبا عليك البحث في توثيق الكود لمعرفة ماذا يمثل توقيع المفوض ثم يجب عليك مطابقته كليا وإن احتجت لاستدعاء ProcessList مع عدة إجراءات ستقوم بإنشاء العديد من الوظائف الخاصة.

دعنا نرى الآن كيف يمكننا استدعاء هذا الإجراء باستخدام تعابير لمدا

```
Class Person
    Public age As Integer
End Class

Sub DoIt()
    Dim list As new List(Of Person)
    'Obtain list of Person from a database, for example
```

```
    ProcessList(list, Function(person As Person) person.age > 50)
End Sub
```

فباستخدام تعابير لمدا لم يعد هناك حاجة لإنشاء وظيفة خاصة للقيام بمنطق المعالجة حيث يتم تعريف المفوض في النقطة التي سيستخدم فيها وهذا أفضل من تعريفه ضمن وظيفة خاصة في مكان ما وفقدان محليتها باستخدام الطريقة الخاصة وبهذا أنت ترى قوة تعابير لمدا وتسهيلها لعملية قراءة وصيانة الكود الخاص بك

لماذا تم تقديم تعابير لمدا

من أجل دعم استعلامات لينك Linq كان يجب إضافة مجموعة من الإمكانيات الجديدة للغة فيجول بايزيك ومن ضمنها كانت تعابير لمدا. افترض أنه لدينا الاستعلام التالي

```
Dim q = From p In Process.GetProgesses() _
        Where p.PriorityClass = ProcessPriorityClass.High _
        Select P
```

فلكي يتم ترجمة هذا التعبير يجري الكثير من العمل تحت الغطاء فالمترجم سيقوم بالمرور عبر المجموعة Process.GetProcesses ويطبق المرشح الموجود في قسم Where عليها ويعيد قائمة بالعمليات التي تطابق ذلك الشرط كما نلاحظ وجود تعبير فيجول بايزيك داخل قسم Where هو p.PriorityClass = ProcessPriorityClass.High وذلك لتطبيق المرشح وهنا يقوم المترجم بإنشاء تعبير لمدا من أجل المرشح الموجود في قسم Where ويطبقه على كل عنصر في قائمة العمليات

```
Dim q = Process.GetProcesses().Where( _
    Function(p) p.PriorityClass = ProcessPriorityClass.High)
```

وأساسا يشكل التعبير لمدا اختصارا للمترجم من أجل اختصار عملية إنشاء الطرق وربطها مع المفوضات حيث يقوم بكل ذلك من أجلك والفائدة التي نجنيها من تعابير لمدا ولا نجنيها عند استخدام الوظائف والمفوضات هي أن المترجم هنا يستخدم الاستدلال المحلي على النوع على تعابير لمدا ففي المثال السابق يتم تحديد نوع المحدد p بناء على الاستخدام وفي هذه الحالة يحدد التعبير في قسم Where تعبير لمدا ويقوم المترجم بالاستدلال آليا على نوع القيمة المعادة من التعبير لمدا بحيث تعتبر ميزة الاستدلال المحلي على النوع المدعومة من قبل المعالج من الإضافات القوية لفيجول بايزيك

الاستدلال المحلي على النوع

تقديم ميزة الاستدلال المحلي على النوع القوية يعني أنه لم يعد عليك أن تقلق حول تحديد النوع الملائم لكل متغير وبالتالي فهي تمكنك من القيام بالعديد من الأمور التي كانت تبدو مستحيلة فالاستدلال على النوع المعاد من تعابير لمدا مفيد جدا فإن كان لديك نوع مفوض تريد ربطه مع تعبير لمدا لم يعد عليك تحديد نوع جميع المحددات

```
Dim lambda As Func(Of Integer, Integer) = Function(x) x * x
```

ففي هذا المثال يكون نوع تعبير لمدا هو Func(Of Integer, Integer) وهو مفوض يأخذ محدد من النوع Integer ويعيد محدد من النوع Integer وكنتيجة لهذا فالمترجم يستدل آليا على أن المحدد x العائد لتعبير لمدا هو من النوع Integer والقيمة المعادة من التعبير لمدا هي Integer أيضا كما يمكنك الاستفادة من الاستدلال على نوع تعابير لمدا عندما تستدعي طريقة تأخذ مفوضا لاحظ الكود التالي

```
Delegate Function ShouldProcess(Of T) (element As T) As Boolean

Sun ProcessList(Of T) (_
            Elements As List(Of T), shouldProcess As ShouldProcess(Of T))
    ' Method body removed for brevity
End Sub
```

في هذه الحالة تأخذ الوظيفة ProcessList تعبير لمدا ويمكن استدعاؤها على الشكل

```
Sub DoIt()
    Dim list As new List(Of A)
    ' fill or obtain elements in list
    ProcessList(list, Function(a) a.x > 50)
```

```
End Sub
```

لاحظ أننا لم نحدد نوع المحدد الممرر للتعبير لمدا كما فعلنا سابقا وذلك لأن المعالج يستدل عليه بنفسه.

كيف يمكن حدوث شئ كهذا؟ في الحقيقة هناك عدة مستويات من الاستدلال على النوع في هذا المثال ففي البداية يرى المترجم ProcessList كإجراء عادي يأخذ list(Of T) كدخل له و ShouldProcess(Of T) في استدعاء ProcessList ويرى المترجم أن list هي المحدد الأول وأنها list(Of Person) وبما أن المحدد الثاني لا يوفر تلميحات حول ماهية نوع T فيقرر المترجم أن T من النوع Person ويستدل من هذا على أن محدد ShouldProcess(Of T) هو من النوع Person وبهذا يستدل على أن المحدد الثاني هو من النوع ShouldProcess(Of T) وأخيرا بما أن تعبير لمدا لا يقدم نوع المحدد الخاص به والمترجم يعرف أن نوع المحدد يعتمد على توقيع
المفوض ShouldProcess(Of T) وقد استدل على أن نوع المحدد a هو Person ويعتبر هذا نوعا قويا من الاستدلال على النوع فليس عليك معرفة نوع محددات المفوض عندما تبني تعبير لمدا وفي الحقيقة من الأفضل ترك المترجم يقوم بذلك العمل نيابة عنك والاستدلال على نوع النتيجة بهذه الطريقة مفيد حقيقة إن لم يكن لديك نوع مفوض وتريد من المترجم أن يقوم بتصنيعه من أجلك علما بأن هذه الميزة متوفرة في فيجول بايزيك فقط

```
Dim lambda = Function(x As Integer) x * x
```

ففي المثال السابق بما أن المحدد x هو من النوع Integer فالمترجم يستدل آليا على أن القيمة المعادة هي من النوع Integer أيضا كنتيجة المعادلة الموجودة في التعبير وبما أن تعبير لمدا لا يمتلك نوعا لهذا يقوم المترجم بتصنيع مفوض مجهول يطابق شكل تعبير لمدا ويربط ذلك النوع المفوض بتعبير لمدا. وهذه ميزة عظيمة لأنها تعني أنه يمكنك إنشاء تعابير لمدا بسرعة بدون أن تحتاج لتعريف الأنواع المفوضة الخاصة بها. فكم مرة كنت في وضع تحتاج فيه لتطبيق مجموعة من المتغيرات وتحتاج إلى فعل ذلك في العديد من الأماكن ففي الكود التالي مرت عدة حالات مشابهة وعادة يمكننا معالجة ذلك بحيث يمكن التحقق من الشرط في مكان واحد بدلا من التشتت في أرجاء الوظيفة

```
Class Motorcycle
    Public color As String
    Public CC As Integer
    Public weight As Integer
End Class

Sub PrintReport(motorcycle As New Motorcycle)
    If motorcycle.color = "Red" And motorcycle.CC = 600 And _
        Motorcycle.weight > 300 And Motorcycle.weight < 400 Then

        ' do something here
    End If

    ' do something here

    If motorcycle.color = "Red" And motorcycle.CC = 600 And _
        Motorcycle.weight > 300 And Motorcycle.weight < 400 Then

        ' do something here
    End If
End Sub
```

وفي بعض الأحيان يستخدم هذا التحقق في هذه الوظيفة فقط ويمكننا إضافة إجراء في الفئة لدعم تلك الوظيفة فقط والقيام بذلك يؤثر على عملية صيانة الكود فماذا لو قام أحد ما باستدعاء هذه الوظيفة في مكان آخر واحتجت للقيام بتعديل ما وقد يؤدي هذا في بعض الفئات إلى وجود وظائف يصعب تعقبها جاعلا خاصية IntelliSense أقل فائدة لوجود العديد من المدخلات الإضافية فيها إضافة إلى خرق منطق المحلية وإن قمنا بذلك باستخدام طريقة منفصلة مختلفة عندها يفضل أن تكون قريبة من الطريقة التي تستخدمها ومع وجود العديد من الأشخاص يعملون على نفس المشروع يصبح من الصعب صيانة المحلية على المدى الطويل وهنا يأتي استخدام تعابير لمدا وترك المترجم يقوم آليا بإنشاء المفوضات ويقوم باستخدامها عند الحاجة

```
Sub PrintReport(motorcycle As New Motorcycle)
    Dim check = Function(m As Motorcycle) m.color =  "Red" And _
```

```
                                m.CC = 600 And _
                                m.weight > 300 And _
                                m.weight < 400

    If check(motorcycle) Then
        ' do something here
    End If

    ' do something here

    If check(motorcycle) Then
        ' do something here
    End If
End Sub
```

قمنا هنا بتعديل منطق تفحص بعض شروط Motorcycle ليستخدم تعابير لمدا عوضا عن سيئات الطرائق الخاصة حيث سيقوم المترجم تلقائيا بإنشاء النوع المفوض ويقوم بالعمل لكي نستطيع استدعاء تعابير لمدا أينما احتاج ذلك وهذه الطريقة مفيدة لأنها تضع المنطق قريب من التصريح حيث نقوم بتصنيع نسخة واحدة ويقوم المترجم بعدها بمعظم عمليات الصيانة ويعتبر هذا مفيدا لأنه يمكنك من بناء تعبير معقد كجسم لتعبير لمدا وباستخدام الربط المتأخر والاستدلال على النوع في هذا السيناريو فلا نحدد نوع تعبير لمدا أو المتغير

```
Dim lambda = Function(x) x * x
```

وهنا أيضا يولد المعالج مفوض مجهول من أجلك ولكن يحدد نوع تعبير لمدا كـ System.Object وهذا يعني أنه قد تم تفعيل الربط المتأخر في هذا السيناريو عندما يكون الخيار Option Strict على الوضع Off ويعتبر هذا السيناريو جيدا بالنسبة لأولئك الذين يعتمدون على الربط المتأخر حيث أن تعابير لمدا تدعم عمليات الربط المتأخر بشكل كامل ففي المثال السابق طالما أن المعامل * معرف على الأنواع الممررة إلى تعبير لمدا فسوف يعمل

```
Dim a = lambda(10)
Dim b = lambda(CDec(10))
Dim c = lambda("This will throw an exception because " & _
                "strings don't support the * operator")
```

وكما ترى من المثال السابق طالما أن المعامل * موجود في مكتبات زمن التشغيل بالنسبة للنوع الممرر فسوف يجري كل شئ بشكل جيد كما أن تعابير لمدا تتأقلم بشكل رائع مع الربط المتأخر في فيجول بايزيك.

الكود المولد تحت الغطاء

بعدما استكشفنا تعابير لمدا دعنا نلقي نظرة على الكود الذي يتم توليده من قبل المترجم. انظر للكود السابق

```
Sub TestLambda()
    Dim doubleIt As Func(Of Integer, Integer) = _
            Function(x As Integer) X * 2
    Console.WriteLine(doubleIt(10))
End Sub
```

أنت تعلم أن Func هو مفوض والمفوضات هي مؤشرات للوظائف فكيف يقوم المترجم إذا بالعمل؟ في هذه الحالة يقوم المترجم بإصدار وظيفة جديدة ويربطها بمفوض يشير إلى تلك الوظيفة الجديدة

```
Private Function $GeneratedFunction$(x As Integer) As Integer
    Return x * 2
End Function

Sub TestLambda()
    Dim doubleIt As Func(Of Integer, Integer) = _
        AddressOf $GeneratedFunction$
```

```
    Console.WriteLine(doubleIt(10)
End Sub
```

حيث يأخذ المترجم تعبير لمدا وينشئ وظيفة جديدة بمحتوياته ويغير عبارة التصريح بحيث يأخذ تعبير لمدا عنوان الوظيفة الجديدة المولدة ففي هذه الحالة يتم توليد الوظيفة بنفس الأب الذي يحتوي على الطريقة التي تستخدم تعبير لمدا فإن كان TestLambda معرف في الفئة C فسوف يتم تعريف الوظيفة الجديدة في الفئة C أيضا ونلاحظ أن هذه الوظيفة غير قابلة للاستدعاء ويتم التصريح عنها باستخدام محدد الوصول Private

تعابير لمدا ورفع المتغيرات

في الأمثلة السابقة يشير جسم تعابير لمدا إلى متغيرات يتم تمريرها إلى تلك المتغيرات ومع ذلك تأتي قوة تعابير لمدا مع ثمار رفع المتغيرات وجميع تعابير لمدا مبنية على مبدأ متشابه. وتعبير لمدا يمكن أن يستخدم متغيرات مرتبطة أو متغيرات حرة لم يتم تعريفها ضمن التوقيع الخاص بتعبير لمدا فالمتغيرات الحرة ممكن أن يكون قد تم التصريح عنها في الإجراء المستدعي للتعبير فقد تكون متغيرات محلية أو محددات ممررة لذلك الإجراء والتعابير المرتبطة تكون تلك التي تم التصريح عنها في جسم التعبير أو عناصر في الفئة المحتوية للتعبير لمدا متضمنا الفئة الأب لتلك الفئة. وهذا هام من أجل التمييز بين المتغيرات المرتبطة والحرة في تعابير لمدا الخاصة بك لأنها تؤثر على دلالة تعبير لمدا والكود الذي يتم توليده وبالتالي يؤثر على صحة برنامجك وهذا مثال يحتوي على تعابير لمدا تستخدم متغيرات مرتبطة وأخرى حرة

```
Function MakeLambda() As Func(Of Integer, Integer)
    Dim y As Integer = 10
    Dim addTen As Func(Of Integer, Integer) = Function(ByVal x) x + y
    Return addTen
End Function

Sub UseLambda()
    Dim addTen = MakeLambda()
    Consloe.WriteLine(addTen(5))
End Sub
```

فهذا الكود سيقوم بطباعة 15 على نافذة الكونسول عندما يتم استدعاء UseLambda ولكن يمكن أن تسأل نفسك كيف يعمل هذا؟ تحدد الوظيفة MakeLambda المتغير y كمتغير محلي والتعبير لمدا يستخدم y ولكن التعبير لمدا يتم إعادته كنوع معاد من الوظيفة MakeLambda والوظيفة UseLambda تحصل على التعبير لمدا من الوظيفة MakeLambda وتنفذ التعبير لمدا ويبدو الأمر كما لو أن المتغير y قد تم تذكره من قبل التعبير لمدا. ففترة حياة المتغير y تنتهي مع نهاية الطريقة MakeLambda فعندما نحصل على التعبير لمدا من MakeLambda فسوف تصبح MakeLambda خارج المجال ويجب إزالة المساحة التي تحجزها في المكدس وبطريقة ما يعلق هذا المتغير مع تعبير لمدا وهذا ما يعرف برفع المتغير Variable Lifting ففي هذه الحالة يدعى المتغير y بالمتغير المرفوع وكما ترى فالمتغيرات المرفوعة تعتبر ميزة برمجية قوية فالمترجم يقوم بالكثير من العمل من أجل تمكينك من إمساك حالة المتغير حيث يحفظها خارج مجال فترة حياتها الطبيعية فعندما يصادف المترجم تعابير لمدا تستخدم متغيرات حرة يقوم برفع المتغير إلى فئة تدعى Closure بحيث تكون فترة حياة هذه الفئة تمتد إلى ما بعد فترة حياة المتغيرات الحرة المستضافة داخلها ويقوم المترجم بإعادة كتابة الوصول إلى المتغيرات في الطرق ليتم الوصول إلى نسختها الموجودة في الفئة Closure

دعنا نسير مرة أخرى عبر المثال MakeLambda

```
Dim MakeLambda() As Func(Of Integer, Integer)
    Dim y As Integer = 10
    Dim addTen As Func(Of Integer, Integer) = Function(ByVal x) x + y
        Return addTen
End Function
```

وكما قمنا بالتحليل سابقا فالمتغير x مرتبط بمحدد التعبير لمدا ولكن المتغير y تعبير حر ويقوم المترجم بالكشف عن ذلك ويتابع بإنشاء الفئة Closure التي تلتقط المتغيرات الحرة كما في تعريف تعبير لمدا

```
Public Class _Closure$__1
    Public y As Integer
```

```
    Public Function _Lambda$__1(ByVal x As Integer) As Integer
        Return x + Me.y
    End Function
End Class
```

يمكنك رؤية أن متغير Closure يلتقط المتغير y ويخزنه في الفئة Closure ويتم تحويل المتغير الحر بعدها إلى متغير مرتبط داخل الفئة Closure كما يقوم المترجم بإعادة كتابة الطريقة التي تحتوي على التعبير لمدا لتبدو كما يلي

```
Function MakeLambda() As Func(Of Integer, Integer)
    Dim Closure As New _Closure$__1
    Closure.y = 10
    Return AddressOf Closure._Lambda$__1
End Function
```

يمكنك الآن رؤية كيف يقوم المترجم بإنشاء المتغير Closure ويعيد كتابة المتغير y الذي تم رفعه ضمن المتغير Closure ويضبط قيمته ويعيد ببساطة عنوان تعبير لمدا المخزن ضمن الفئة Closure ومن الهام ملاحظة أن المترجم يقوم برفع المتغيرات الحرة في تعابير لمدا فقط ويتم التقاط حالة المتغير في Closure الذي يبقى موجودا طالما أن تعبير لمدا بقي موجودا. انظر للمثال التالي

```
Sub Test()
    Dim y As Integer = 10
    Dim Lambda As Func(Of Integer, Integer) = Function(ByVal x) x + y
    y = 20
    Console.WriteLine(Lambda(5))
End Sub
```

ما هي القيمة التي تظهر عند تنفيذ الوظيفة السابقة؟ إن قلت 25 فقد أصبت. فلماذا 25 إذا؟ المترجم يقوم بالتقاط وإعادة كتابة جميع المتغيرات الحرة y إلى نسخة Closure كالتالي

```
Sub Test()
    Dim Closure As New $Closure_Compiler_Generated_Name$
    Closure.y = 10
    Dim Lambda = AddressOf Closure.Lambda_1
    Closure.y = 20
    Console.WriteLine(Lambda(5))
End Sub
```

ففي الوقت الذي يتم تنفيذ تعبير لمدا فيه تكون قيمة y قد تغيرت إلى 20 وبهذا فعندما يتم تنفيذ تعبير لمدا يعيد 20 + 5 وهذا هام جدا لأنه عندما نأتي للحديث عن الحلقات وأن المتغيرات الحرة يتم التقاطها في Closure وحيد قد ترى تصرفات غريبة. انظر للمثال التالي

```
Sub Test()
    For I = 1 To 5
        StartThread(Function() I + 10)
    Next
End Sub
```

افرض أن StartThread ينشئ مسارا جديدا ويطبع النتيجة على الكونسول وطالما أنه تم التقاطه إلى Closure فيمكن أن تكون الحلقة قد غيرت قيمة I في الوقت الذي يقوم المسار فيه باستدعاء تعبير لمدا وفي هذه الحالة فالبرنامج قد لا يطبع النتيجة المتوقعة وبدلا عن ذلك عليك رؤية المتغير الملتقط داخل الحلقة

```
Sub Test
    For I = 1 To 5
        Dim x = I
        StartThread(Function() x + 10)
    Next
```

```
End Sub
```

فالكود سيلتقط الآن قيمة x في Closure والبرنامج سيطبع القيم كما هو متوقع ومن الهام جدا معرفة أية متغيرات سيتم رفعها عندما سيتم تنفيذ تعبير لمدا ومتى سيتم تغيير قيمة تلك المتغيرات المرفوعة وبذلك يمكنك التأكد من أن برنامجك يتم تنفيذه بصورة صحيحة.

استخدام تعابير لمدا بالشكل الأمثل

في فيجول بايزيك 2008يمكنك تمرير تعبير واحد كجسم لتعبير لمدا وقد تم تقديم كلمة محجوزة ثلاثية جديدة هي If لتمكنك من كتابة تعابير شرطية ذات نوع كامل

```
Dim x = IF(condition, 10, 20)
```

والكلمة المحجوزة If مشابهة لاستدعاء الوظيفة IIF فيما عدا أنها آمنة ضد النوع. وهذا يعني أنه في المثال السابق يتتبع المترجم كلا فرعي الكلمة المحجوز If ويعيد Integer وبهذا فهو يطبق قواعد الاستدلال على النوع ويقرر أن نوع x هو integer ولكن استخدام IIf سيعيد النوع Object. كما يمكنك استخدام If في تعبير لمدا

```
Dim x = Function(c As Customer) _
    If(c.Age >= 18, c.Address, c.Parent.Address)
```

ففي المثال السابق افترض أنه لديك فئة Customer يتضمن تعريفها الخاصية Address التي تمثل العنوان الحالي للزبون حيث أن تعبير لمدا يستخدم التعبير الثلاثي Ternary Expression لتطبيق الشرط على محدد الدخل فإن كان عمر الزبون مساويا أو أكثر من 18 فهو يعيد عنوانه وإلا فهو يعيد عنوان والده وهنا يتم استخدام الاستدلال على النوع أيضا ويقوم المترجم بتحديد نوع تعبير لمدا ليكون Address ثم يقوم بإنشاء النوع المفوض x بالطريقة التي تمت مناقشتها سابقا حيث يأخذ النوع المفوض Customer كدخل ويعيد Address.

Object Initializers

تمكنك Object Initializers من تحديد خصائص غرض معقد ضمن تعبير واحد وتستخدم لتعريف متغيرات من كلا من الأنواع المعروفة والمجهولة فلو فرضنا أنه لدينا فئة بسيطة Employee معرفة على الشكل

```
Public Class Employee
    Private _name As String
    Private _Salalry As Short
    Private _Address As String

    Public Property Name() As String
        Get
            Return _name
        End Get
        Set(ByVal value As String)
            _name = value
        End Set
    End Property

    Public Property Salary() As Short
        Get
            Return _Salalry
        End Get
        Set(ByVal value As Short)
            If value > 0 Then
                _Salalry = value
            End If
        End Set
    End Property

    Public Property Address() As String
        Get
            Return _Address
        End Get
        Set(ByVal value As String)
            _Address = value
        End Set
    End Property
End Class
```

يمكننا باستخدام تعريف متغير يشير إلى تلك الفئة واسندا الخصائص كما في الكود التالي مع أننا لسنا مضطرين هنا لضبط قيم كافة الخصائص التي تحتويها الفئة فنقوم بضبط قيم الخصائص التي نحتاج لضبطها فقط

```
Dim Empl3 = New Employee With {.Name = "Mazen", .Salary = 8500}

Dim Empl1 As New Employee With {.Name = "Reem", .Salary = 10000}
```

كما يمكننا اختصار قسم AS هنا فيمكن كتابة التصريح كما يلي وذلك اعتماد على local type inference

```
Dim Empl5 = New Employee With {.Name = "Ahmad"}
```

بينما كنا في السابق وباستخدام نفس الفئة كما يلي

```
        Dim Empl2 As New Employee
        With Empl2
            .Name = "Ahamd"
            .Salary = 11500
        End With
```

وإن كانت لدينا فئة تحتاج لتمرير قيم لمشيد الفئة مثل الفئة Person مثلا فيمكننا أيضا استخدام نفس الطريقة لضبط خصائص أخرى لا يتم تمريرها لمشيد الفئة

```
Dim Per1 As New Person("Ghassan") With {.Address = "Damas"}
```

كما تستخدم هذه الطريقة أيضا لتعريف الأنواع المجهولة

```
Dim Visitor = New With {.Name = "Mussa", .Account = 232536}
```

وكما نلاحظ من طريقة التعريف فصيغة تعريف الأنواع المعروفة مماثلة في الشكل للأنواع المجهولة ففي الأنواع المعروفة لاحظ وجود اسم الفئة بعد الكلمة new بينما عندما نعرف نوعا مجهولا لايوجد اسم للفئة بعد الكلمة new بسبب أن الأنواع المجهولة ليس لها اسم فئة قابلة للاستخدام فعند استخدام فئة معروفة عند التصريح يجب أن تكون الخصائص التي نريد ضبط قيمها موجودة فعلا والتصريح ينشئ متغيرا يشير إلى تلك الفئة ومن أجل تعريف النوع المجهول يقوم المترجم بإنشاء فئة جديدة لذلك المتغير تحتوي الخصائص المشار إليها في التصريح ويحدد اسمها عند الترجمة وقد يختلف لاسم من عملية ترجمة لأخرى لذلك لا يمكن الاعتماد على اسم الفئات المجهولة ضمن الكود أو التعريف

وإليك بعض الملاحظات الخاصة بالتعريف

- قائمة التعريف بعد With لا يمكن أن تكون فارغة

- لا يمكن تكرار تعريف قيمة لخاصية أكثر من مرة في نفس التعريف

- يمكن ضبط قيمة خاصية من خاصية أخرى

- في حال كانت إحدى الخصائص فئة يمكن تعشيش التصريح بنفس الطريقة

```
Dim cust12 = New Customer With {.Name = "Toni Poe", _
                                .Address = New AddressClass _
                                With {.City = "Louisville", _
                                      .State = "Kentucky"}}
Console.WriteLine(cust12.Address.State)
```

- لا يمكن استخدام عناصر مشتركة Shared أو للقراءة فقط ReadOnly أو الثوابت أو استدعاء الطرق في القائمة بعد كلمة With

- لا يمكن استخدام الخصائص التي تمتلك فهرسا أو المشروطة كمصفوفة مثلا فالتعريفات التالية مثلا غير صحيحة

```
'' Not valid.
' Dim c1 = New Customer With {.OrderNumbers(0) = 148662}
' Dim c2 = New Customer with {.Address.City = "Springfield"}
```

ترقية مشاريع 2005 لتعمل على 2008 ثم إضافة دعم Linq لتلك المشاريع

- افتح مشروعك ضمن بيئة تطوير 2008 فيظهر لك معالج الترقية تلقائيا – اضغط next
- يظهر لك المعالج خيار عمل نسخة احتياطية للملفات القديمة فنختار الخيار Yes, create a backup before converting ثم يقترح مكانا لوضع النسخة الاحتياطية فيه في مربع النصوص تحت الكلمة Location for backup حيث يمكنك تغييره بالضغط على الزر Browse أو تركه كما هو - اضغط next فيظهر لك معلومات عن الترقية - اضغط هنا Finish للبدء بعملية الترقية
- بعد الانتهاء تظهر لك نافذة تخبرك بانتهاء عملية الترقية وفيها خيار Show the conversion log when the wizard is closed وبتفعيل هذا الخيار يظهر لك تقرير عن عملية التحويل بعد إغلاق المعالج هنا اضغط Close
- كما تجدر ملاحظة أن مشروعنا بعد التحويل حتى هذه النقطة مازال متوافقا مع بيئة تطوير الـ2005
- من Solution Explorer انقر بزر الفأرة اليساري نقرا مزدوجا على My Project لفتح خصائصه
- افتح صفحة Compile وقم بضبط الخيار Option Infer إلى On الذي يخبر المعالج أن يستدل على نوع المتغيرات المحلية من التعبير الذي يضبط قيمة ذلك المتغير إن لم نزوده بنوع ذلك المتغير - ولمزيد من المعلومات حول هذا الخيار يمكنك قراءة موضوعي في المنتدى بعنوان الاستدلال المحلي على النوع Local Type Inference
- انتقل لأسفل صفحة Compile ثم اضغط الزر Advanced Compile Options الذي يظهر لنا صندوق حوار بخيارات الترجمة المتقدمة حيث نرى في أسفل هذه النافذة الخيار Target framework والذي مازال مضبوطا على الـ Framework 2.0 حتى الآن حيث يمكننا هذا الخيار من كتابة كود يعمل على أي نسخة من نسخ الفريمورك الموجودة ضمن بيئة تطوير واحدة فبدلا من تنصيب عدة نسخ من Visual Studio على نفس الجهاز بهدف العمل مع أكثر من Framework أصبح الآن بإمكانك عمل ذلك من داخل بيئة تطوير واحدة هي 2008 ومن أجل تمكين Linq سنختار Framework 3.5 ثم اضغط على Ok فيظهر لنا تحذير بأنه سيتم إغلاق وفتح المشروع تلقائيا وأنه سيقوم بحفظ أية تغييرات غير محفوظة تلقائيا وهنا اضغط Yes
- انقر بزر الفأرة اليساري نقرا مزدوجا على My Project لفتح خصائصه وانتقل للصفحة References فنلاحظ أنه قد تم إضافة مرجعا تلقائيا للمكتبة system.core.dll من الـ Framework 3.5 للمشروع وذلك لأننا قمنا بترقية Target Framework ومن أجل تمكين استعلامات Linq علينا إضافة بعض المراجع واستيراد بعض مجالات الأسماء اعتمادا على مزود Linq الذي نحتاج لاستخدامه
- فمن أجل إضافة مرجع لـ Linq to Object نختار من القائمة أسفل Import Namespaces المجال System.linq الذي يمكننا من كتابة استعلامات Linq على الأغراض المختلفة حيث أصبح بإمكاننا كتابة الاستعلام التالي للحصول على أسماء الملفات في المجلد الحالي

```
Dim MyFiles = From Files In My.Computer.FileSystem.GetFiles(CurDir()) _
              Select Files
```

ونلاحظ أنه بسبب خاصية Option Infer المضبوطة إلى On أن المترجم قد ضبط نوع المتغير MyFiles إلى IEnumerable(Of String) وذلك عند تمرير مشيرة الفأرة فوق المتغير MyFiles

- ومن أجل تمكين استعلامات Linq للاستعلام من الـ Datasets سنحتاج لبعض الإجراءات الإضافية وهنا افتح خصائص My Project وعد للصفحة References واضغط الزر Add ومن صفحة .net أضف مرجعا لـ System.Data.DataSetExtensions ثم اضغط OK
- سنحتاج الآن لإعادة توليد الـ Dataset التي نريد استخدامها مع استعلامات Linq ولعمل ذلك ننقر بزر الفأرة اليميني على الـ Dataset المناسبة ثم نختار Run Custom Tool من القائمة وهنا تقوم بيئة التطوير بإعادة كتابة كود الـ Dataset لتصبح الجداول ضمنها موروثة من فئة داعمة لـ Linq تسمى TypedTableBase وهذا الذي يستدعي الحاجة لإضافة مجال الأسماء System.Data.DataSetExtensions وأصبح بإمكاننا الآن كتابة استعلامات من الـ dataset مثل الاستعلام

```
Dim MyCats = From category In Me.CategoryProductDataSet.Categories _
             Where category.CategoryName Like "C*" _
             Select category
```

ونلاحظ هنا أيضا أن المترجم قد ضبط نوع المتغير MyCats بناء على جملة الاستعلام المرتبطة به تلقائيا إلى EnumerableRowCollection

- ولكتابة استعلامات Linq to XML عد إلى صفحة References في خصائص MyProject واضغط الزر Add وأضف مرجعا لـ System.Xml.Linq ثم انتقل إلى القائمة Imported namespaces وقم باختيار System.Xml.Linq حتى نتمكن من كتابة استعلامات من XML

- كما يوجد مزود آخر ربما نريد استخدامه وهو مزود Linq to Sql حيث يمكن إضافته بسهولة فقط قم باختيار Add New Item من قائمة Project وقم بإضافة Linq to Sql Classes وهذا سيقوم تلقائيا بإضافة المراجع والاستيرادات المناسبة لمشروعنا حيث سنلاحظ من صفحة References في خصائص MyProject أنه قد تم إضافة مرجعا لـ System.Data.Linq

أصبح الآن بإمكاننا استخدام Linq للاستعلام على الأغراض Objects المختلفة في مشروعنا بالإضافة إلى الاستعلام من Dataset أو XML أو حتى Sql Databse

Linq To Object وأساسيات استعلامات Linq

باستخدام مزود Linq to Object يمكننا الاستعلام من أغراض دوت نيت المختلفة طالما هي تدعم الواجهة IEnumerable أو الواجهة IEnumerable(T) فمثلا يمكننا كتابة استعلام للحصول على قائمة بالملفات الموجودة في المجلد الحالي

```
Dim Files = From Fi In My.Computer.FileSystem.GetFiles(CurDir()) _
            Order By Fi
```

نلاحظ بداية أننا عندما قمنا بالتصريح عن المتغير Files في بداية الاستعلام لم نصرح عن نوع المتغير وذلك لأن المترجم هنا يستدل على نوعه من التعبير الذي يضبط قيمته وهنا أفترض أنك قد درست موضوعي بخصوص الاستدلال المحلي عن النوع وفي حالة الاستعلام السابق إن قمت بتمرير مؤشر الفأرة فوق المتغير Files ستجد أن بيئة التطوير قامت بضبط نوعه إلى System.Linq.IOrderedEnumeable(Of String) باستخدام الاستدلال المحلي على النوع ونبدأ بكتابة الاستعلام بالقسم From حيث نحدد أنه لدينا متغير Fi يحصل على قيمته من الكائن الذي يلي الكلمة In وفي استعلامنا هنا My.Computer.FileSystem.GetFiles(CurDir()) حيث تجدر الملاحظة إلى أن استعلامات Linq جميعها تكون سطرا واحدا في لغة فيجول بايزيك لذا من أجل التنسيق وقابلية القراءة والتعديل نقسم العبارة على عدة أسطر باستخدام محرف المتابعة _ ونريد هنا أن نخرج قائمة مرتبة بأسماء الملفات لذا نستخدم قسم OrderBy ليقوم بترتيب أسماء الملفات المعادة من الاستعلام وكي نظهر قائمة الملفات هذه في ListBox نستخدم حلقة For … Each للدوران عبر عناصر المجموعة المعادة من الاستعلام بما أنها تحقق الواجهة IEnumerable وإضافتها عنصرا عنصرا لمربع القائمة كما في المثال

```
For Each f In Files
    Me.ListBox1.Items.Add(f)
Next
```

كما يمكننا استخدام الاستعلام ضمن استعلام آخر فللحصول على معلومات الملفات المعادة من الاستعلام السابق يمكننا كتابة الاستعلام التالي حيث يحدد القسم Select أن النتيجة المعادة هي مجموعة من FileInfo بحسب القيمة المعادة من الدالة GetFileInfo

```
Dim FInfo = From File In Files _
            Select My.Computer.FileSystem.GetFileInfo(File)
```

ويمكننا إظهار نتيجة هذا الاستعلام في DataGridView مباشرة وذلك بضبط قيمة الخاصية DataSource إلى نتيجة عائد الدالة ToList الخاصة بمتغير الاستعلام FileInfo كما في الكود

```
Me.DataGridView1.DataSource = FInfo.ToList
```

كما يمكننا استخدام قسم Select لتخصيص المعلومات المعادة من الاستعلام وبشكل مشابه لما كنا نفعله في عبارة Select التي نستخدمها في استعلامات SQL فبدلا من إعادة كافة خصائص FileInfo كما في الاستعلام السابق يمكننا كتابة استعلامنا لإظهار اسم الملف ووقت الإنشاء فقط كما في المثال

```
Dim MyInfo = From Fi In FInfo _
             Select Fi.Name, Fi.CreationTime
```

كما يمكننا إعادة نتيجة هذا الاستعلام ضمن غرض من إنشائنا بدلا من النوع الذي يتم تحديده تلقائيا كنتيجة للاستعلام فإن كان لدينا فئة بسيطة باسم MyFiles تمتلك خاصيتين CreationTime من النوع Date و Name من النوع String يمكننا عندها إعادة كتابة استعلامنا بالشكل

```
Dim MyFiles = From Fi In FInfo _
           Select New MyFiles() With {.Name = Fi.Name, .CreationTime = Fi.CreationTime}
```

حيث سيعيد الاستعلام النتيجة كمجموعة IEnumerable(Of MyFiles) ويمكنك مراجعة موضوعي Object Initializers بخصوص صيغة تعريف الفئة MyFiles ضمن الاستعلام

وإن أردنا تخصيص ناتج الاستعلام للحصول على الملفات التي تحمل الامتداد exe فقط مثلا نستخدم قسم where الذي يحدد شرط الانتقاء في جملة الاستعلام مستخدمين الطريقة EndsWith لاختيار اسم الملف الذي ينتهي بـ .exe عندها يمكننا كتابة الاستعلام بالشكل التالي

```
Dim ExeFiles = From Fi In My.Computer.FileSystem.GetFiles(CurDir()) _
               Where Fi.EndsWith(".exe") _
               Select My.Computer.FileSystem.GetFileInfo(Fi)
```

وإن أردنا الحصول على مجموع حجوم الملفات من النوع exe يمكننا استخدام ناتج الاستعلام السابق في استعلام جديد

```
Dim ExeSize = Aggregate Fs In ExeFiles _
              Into Sum(Fs.Length)

Me.TextBox1.Text = ExeSize
```

فهنا استخدمنا Aggregate بدلا من From في بداية الاستعلام عندما نريد استخدام الدالات التجميعية للحصول على نتائج تجميعية من الاستعلام ففي قسم Into استخدمنا الدالة Sum للحصول على مجموع الحجوم

إذا افترضنا أنه لدينا فئة باسم Personnel تمتلك الخصائص Name و Birthdate و Age و City و Salary وقمنا بإنشاء قائمة تحتوي على مجموعة عناصر تمتلك نوع هذه الفئات

```
Dim Pers As New List(Of Personnel)
```

وبافتراض أن هذه القائمة تحتوي على العديد من العناصر يمكننا كتابة مجموعة من الاستعلامات للحصول على معلومات مختلفة حول عناصر هذه القائمة فإن أردنا قائمة بالأشخاص الذين رواتبهم أكثر من 10000 وأردنا في الناتج فقط الاسم والعمر والراتب وترتيب النتائج بحسب الراتب يمكننا كتابة الاستعلام كما يلي

```
Dim Prs = From p In Pers _
          Where p.Salary > 10000 _
          Order By p.Salary _
          Select p.Name, p.Age, p.Salary
```

وإن أردنا فقط الأشخاص الذين يقيمون في مدينة دمشق فقط سيصبح استعلامنا بالشكل

```
Dim Prs = From p In Pers _
          Where p.Salary > 10000 And p.City = "Damascus" _
          Order By p.Salary _
          Select p.Name, p.Age, p.Salary
```

وإذا أردنا الأشخاص في مدينتي دمشق وحماة الذين رواتبهم أكثر من10000 يصبح استعلامنا كما يلي

```
Dim Prs = From p In Pers _
          Where p.Salary > 10000 _
          And (p.City = "Damascus" Or p.City = "Hama") _
          Order By p.Salary _
          Select p.Name, p.Age, p.Salary, p.City
```

وإن أردنا الحصول على مجموع رواتب الأشخاص المقيمين في حلب يمكننا كتابة الاستعلام

```
Dim prs = Aggregate p In Pers _
          Where p.City = "Aleppo" _
          Into Sum(p.Salary)
```

وإن أردنا الحصول على معلومات الشخص الذي يحصل على اعلى راتب

```
Dim pr = From p In Pers _
         Aggregate pa In Pers _
         Into a = Max(pa.Salary) _
         Where p.Salary = a _
         Select p
```

وإن أردنا معلومات من يحصل على أقل راتب في مدينة حلب

```
Dim pr = From p In Pers _
         Aggregate pa In Pers _
         Where pa.City = "Aleppo" _
         Into a = Min(pa.Salary) _
         Where p.Salary = a And p.City = "Aleppo" _
         Select p
```

وبافتراض انه لدينا فئة ثانية باسم Branches تمتلك الخصائص BranchName و City وقمنا بإنشاء قائمة تحتوي على مجموعة عناصر من نوع هذه الفئات

```
Dim Brnch As New List(Of Branches)
```

يمكننا كتابة الاستعلام التالي للحصول على اسم الشخص والفروع المتوفرة في مدينته

```
Dim BrnPer = From pe In Pers _
             Join br In Brnch On pe.City Equals br.City _
             Select PersonName = pe.Name, BranchName = br.BranchName, pe.City
```

حيث استخدمنا join لربط مجموعة الأشخاص مع مجموعة الفروع باستخدام المدينة ثم استخدمنا Select لتحديد الحقول المطلوب إخراجها في نتيجة الاستعلام

الآن نريد إظهار قائمة بجميع الأشخاص مع الفروع المتوفرة لهم وبذلك سيصبح استعلامنا بالشكل

```
Dim PreBr = From pe In Pers _
            Group Join br In Brnch On pe.City Equals br.City _
            Into AvBr = Group _
            Select pe, AvBr
```

في البداية اخترنا المتغير Pe من قائمة الأشخاص Pers ثم استخدمنا Group Join لربط هذه القائمة مع قائمة الفروع باستخدام حقل المدينة ثم وضعنا ناتج الربط من القائمة الثانية في متغير AvBr في قسم Into ثم حددنا في قسم Select النتائج التي نريدها Pe و AvBr

ويمكننا إظهار ناتج الاستعلام في ListBox باستخدام حلقتي For … Each متداخلتان

```
For Each a In PreBr
    Me.ListBox1.Items.Add(a.pe.Name & ": " & a.pe.City)
    For Each b In a.AvBr
        Me.ListBox1.Items.Add(".... " & b.BranchName)
    Next
Next
```

أو إذا أردنا استخدام DataGridView لإظهار النتائج سنحتاج إلى تحكمان DataGridView و تحكمان BindingSource ولعمل ذلك في البداية سنعرف متغيرا على مستوى النموذج من النوع Dictionary حيث تكون المفاتيح هي الأشخاص والقيم هي الفروع كما يلي

```
Private Gper As Dictionary(Of Personnel, IEnumerable(Of Branches))
```

ثم نقوم بوضع ناتج الاستعلام في متغيرنا Gper حيث نستخدم تعابير لمدا – هل درست المواضيع المتعلقة بتعابير لمدا - لإضافة المفاتيح والقيم إليه كما في الكود

```
Gper = PreBr.ToDictionary(Function(x) x.pe, Function(y) y.AvBr)
```

ثم سنقوم بربط تحكمات BindingSource مع تحكمات DataGridView كما في الكود

```
Me.DataGridView1.DataSource = Me.BindingSource1
Me.DataGridView2.DataSource = Me.BindingSource2
```

ثم نقوم بضبط خاصية DataSource لـ BindingSource1 إلى قيم مفاتيح Gper

```
Me.BindingSource1.DataSource = Gper.Keys
```

وبذلك يتم إظهار قيم المفاتيح في DataGridView1 التي ستظهر قائمة الأشخاص ولإظهار قائمة الفروع المتوفرة لكل شخص في DataGridView1 في DataGridView2 نقوم بمعالجة الحدث CurrentChanged لـ BindingSource1 باستخدام سطر الكود التالي

```
Me.BindingSource2.DataSource = Gper(CType(Me.BindingSource1.Current, Personnel))
```

الذي يحصل على القيمة في الـ Dictionary المقابلة للسجل الحالي في BindingSource1 ويضبطها كـ DataSource لـ BindingSource2 فيتم عرضها في DataGridView2 التي ستظهر قائمة الفروع المتوفرة لذلك الشخص عند النقر عليه في DataGridView1

Linq To DataSet

بافتراض أن قاعدة البيانات Northwind مثبتة في جهازك من نافذة Data Sources في بيئة التطوير أضف مصدر بيانات جديد لمشروعك يتضمن الجدولين Categories و Products وباستخدام طريقة السحب والإفلات اسحب الجدول Categories إلى سطح النافذة الفارغ لإضافة DataGridView مع بعض التحكمات للنموذج ثم من نافذة DataSources وسع العقدة بجانب الجدول Categories واسحب الجدول Products الذي بداخل الجدول Categories إلى سطح النافذة ليتم إنشاء DataGridView ثانية أسفل الأولى خاصة بالجدول Products طبعا لن أقوم بشرح هذه العملية بتفصيل أكثر بما أنها برمجة قواعد بيانات ودورتنا تتحدث عن Linq فإن واجهت مشكلة ابحث في القسم المناسب في المنتدى أو حاول رؤية فيديوهات ميكروسوفت التعليمية بخصوص هذه النقطة وقبل المتابعة يجب أن تتأكد أن مشروعك يعمل جيدا وأن الـ DataGridView الثانية تعرض البيانات المقابلة لما تم اختياره من الأولى فقط.

انتقل إلى محرر الكود للنموذج وقم بإنشاء إجراء معالجة للحدث CurrentChanged للتحكم CategoriesBindingSource وللحصول على السطر الحالي نستخدم الكود التالي

```
Dim row As NorthwindDataSet.CategoriesRow
row = CType(CType(Me.CategoriesBindingSource.Current, _
      DataRowView).Row, NorthwindDataSet.CategoriesRow)
```

وللحصول على إجمالي قيمة البضائع في تلك الفئة والتي مازال إنتاجها مستمرا يمكننا كتابة الاستعلام

```
Dim Total = Aggregate Product In NorthwindDataSet.Products _
            Where Product.CategoryID = row.CategoryID _
            AndAlso Product.Discontinued = False _
            Into Sum(Product.UnitPrice * Product.UnitsInStock)
```

وبافتراض أنك متابع معي منذ البدء أصبحت تعرف طريقة الاستعلام فهنا استخدمنا Aggregate في بداية الاستعلام بما أننا نريد استخدام الدالات التجميعية للحصول على مجموع الكلفة الإجمالية للبضائع التي مازال إنتاجها مستمرا ثم حددنا في قسم Where الشرط بأننا نريد أن يتم تجميع المنتجات الموافقة للفئة المحددة row.CategoryID = Product.CategoryID وأن إنتاجها ما زال مستمرا Product.Discontinued = False ثم استخدمنا قسم Into للحصول على الإجمالي المطلوب وذلك بتمرير جداء قيمة المنتج والعدد الموجود للدالة التجميعية Sum. ولإظهار نتيجة الاستعلام على النموذج ضع صندوق نصوص على النموذج واستخدم الكود التالي لوضع القيمة فيه الذي يستخدم الأمر Format لتنسيق القيمة المعادة من الاستعلام بتنسيق عملة

```
Me.TextBox1.Text = Format(Total, "c")
```

لتنفيذ بعض التصفية على الفئات أضف تحكم صندوق نصوص وزر إلى شريط الأدوات الموجود في أعلى النموذج وفي إجراء حدث النقر على الزر أدخل الاستعلام التالي

```
Dim SelCat = From Cat In Me.NorthwindDataSet.Categories _
             Where Cat.CategoryName.ToLower Like  Me.ToolStripTextBox1.Text.ToLower & "*" _
             Select Cat

Me.CategoriesBindingSource.DataSource = SelCat.AsDataView
```

حيث قمنا باستخدام Like في قسم Where لتصفية النتائج المعادة من الاستعلام تماما كما نفعل في استعلامات قواعد البيانات ثم نقوم بضبط خاصية DataSource لـ CategoriesBindingSource لكي يظهر لنا نتائج الاستعلام حيث أن الطريقة AsDataView الخاصة بالاستعلام تعيد غرض DataView داعم لـ Linq يمثل استعلام Linq to DataSet

دعنا نجري بعض الاستعلامات الأخرى وسع نافذة مشروعك قليلا وأضف DataGridView وزر أوامر جديدان عليها ثم سنستخدم كودنا السابق الذي يحصل على متغير يحمل السجل الحالي في CategoriesBindingSource للحصول على معلومات السجل الحالي الذي تم اختياره في CategoriesDataGridView وذلك في بداية إجراء معالجة الحدث Click لزر الأوامر

```
Dim row As NorthwindDataSet.CategoriesRow
row = CType(CType(Me.CategoriesBindingSource.Current, _
      DataRowView).Row, NorthwindDataSet.CategoriesRow)
```

الآن يمكننا كتابة الاستعلام التالي للحصول على قائمة بالمنتجات التي مازالت قيد الإنتاج مع السعر الإجمالي للموجود منها حاليا وإظهار النتيجة في DataGridView1 والتي من الفئة التي تم اختيارها من الشبكة الخاصة بالفئات

```
Dim Prods = From pr In NorthwindDataSet.Products _
            Where pr.CategoryID = row.CategoryID _
            And pr.Discontinued = False _
            Select pr.ProductName, pr.UnitPrice, _
            pr.UnitsInStock, Total = (pr.UnitPrice * pr.UnitsInStock)

Me.DataGridView1.DataSource = Prods.ToList
```

أضف ثلاث صناديق نصوص إلى النافذة حيث سنقوم بإنشاء استعلام جديد لحساب متوسط سعر الوحدات الموجودة ومجموع الكميات والقيمة الإجمالية وذلك من أجل نفس الفئة التي أظهرنا نتائجها في الاستعلام السابق

```
Dim ProdSums = Aggregate pr In NorthwindDataSet.Products _
               Where pr.CategoryID = row.CategoryID _
               And pr.Discontinued = False _
               Into UntiSum = Sum(pr.UnitsInStock), PriceAvg = Average(pr.UnitPrice), _
               TotalValue = Sum(pr.UnitPrice * pr.UnitsInStock)

Me.TextBox2.Text = ProdSums.PriceAvg.ToString("#,###.00")
Me.TextBox3.Text = ProdSums.UntiSum
Me.TextBox4.Text = ProdSums.TotalValue
```

لاحظ أن طريقة كتابة استعلامات Linq قريبة جدا من طريقة كتابة استعلامات select في SQL مع بعض الاختلاف في الترتيب وأن صيغة هذه الاستعلامات متشابهة مهما اختلف مزود Linq الذي نتعامل معه حيث يمكننا استخدام الأشكال المختلفة للاستعلام التي وردت في الدرس السابق المتعلق بـ Linq to Object هنا أيضا عندما يتعلق الأمر بـ Linq to Dataset وفي الدروس المستقبلية عندما نتحدث عن Linq to xml و Linq to Sql

مثال عملي على Linq To DataSet مع استخدام Lambda Expressions

الهدف من المثال

١. حفظ بيانات DataSet في ملف xml واستعادتها منه والتعامل معها بدون الحاجة لوجود قاعدة بيانات
٢. الاستعلام من الـ DataSet باستخدام Linq ومن أكثر من جدول وإدخال بعض الحسابات في جملة الاستعلام واستخدام عبارة Join لمنع التكرار الخاطئ للبيانات
٣. استخدام Lambda Expressions للقيام بالحسابات من أجلنا والاستفادة من ميزة رفع المتغيرات
٤. إظهار نتيجة استعلام Linq في DataGridView مباشرة

من أجل ترك الموضوع عام وبما أن مجموعة البيانات DataSet يمكن ربطها مع أي قاعدة بيانات سأقوم بالعمل على مجموعة بيانات غير مربوطة مع قاعدة بيانات حيث يمكنك تطبيق الأفكار الواردة هنا على أي قاعدة بيانات يمكن ربطها مع أي DataSet وسيكون مثالي الذي سنسير عليه هنا معتمد على خدمة مصرفية تدعى بالودائع لأجل

افتح أي إصدار من فيجول ستوديو 2008 وأنشئ مشروعا جديدا من نوع Windows Forms Application ثم من قائمة Project اختر الأمر Add New Item وأضف DataSet للمشروع وقم بتسميتها MyDataSet ثم في محرر التصميم الرسومي لمجموعة البيانات انقر بزر الفأرة اليميني واختر الأمر DataTable من قائمة Add في قائمة السياق ثم قم بإعادة تسميته ليصبح اسمه Customers ثم أضف للجدول Customers الحقول التالية مع الخصائص الموضحة بجانب كل منها

الحقل ID الخاصية AutoIncrement بالقيمة True والخاصيتان AutoIncrementSeed و AutoIncrementStep كلتاهما إلى القيمة 1 و نوع البيانات System.Int32 ثم انقر بزر الفأرة اليميني على الحقل ID واختر الأمر Set Primary Key من قائمة السياق لتحديد الحقل كمفتاح أساسي

الحقل CustomerName نوع البيانات System.String و MaxLength بالقيمة 25

الحقل CurrentAccountNumber نوع البيانات System.String و MaxLength 25

حيث أن الحقل ID هو معرف الزبون و CustomerName هو اسم الزبون والحقل CurrentAccountNumber هو رقم الحساب الجاري لدى المصرف

أضف جدول آخر لمجموعة البيانات باسم Wadaeaa وأضف له الحقول التالية مع الخصائص الموضحة بجانب كل منها

الحقل WadeaaNumber بنوع بيانات String

الحقل CustomerID بنوع بيانات Int32

الحقل InterestRate بنوع بيانات Decimal و NullValue مساوية لـ 7.5

الحقل WadeeaPeriod بنوع بيانات Int16 و BullValue مساوية لـ 3

الحقل StartDate بنوع بيانات DateTime

الحقل WadeaaAmount بنوع بيانات Int32 و NullValue بقيمة 10000

حيث WadeaaNumber هو رقم الوديعة و CustomerID هو حقل مرتبط بجدول الزبائن و InterestRate نسبة الفائدة و WadeeaPeriod فترة الوديعة بالأشهر و StartDate تاريخ فتح الوديعة و WadeaaAmount قيمة الوديعة

سنضيف الآن علاقة بين الجدولين: انقر بزر الفأرة اليمني على الجدول Customers ومن القائمة الفرعية Add اختر Relation ثم اضبط Parent Table إلى Customers و Child Table إلى Wadaeaa ثم اختر Key Columns ليضم الحقل ID فقط و Foreign Key Columns ليضم الحقل CustomerID فقط ثم اختر الخيار Both Relations And Foreign Key Constraint ثم اضغط Ok من أجل حفظ العلاقة ثم اختر الأمر Save All من القائمة File

انتقل إلى محرر النماذج الخاص بـ Form1 واجعل مساحة Form1 كبيرة كفاية لتتسع لـ 2 × DataGridView مع شريط أدوات وبعض التحكمات الأخرى التي سنضيفها لاحقا ثم انتقل لنافذة Data Sources واسحب الجدول Customers ثم ألقه على سطح Form1 فيتم إضافة DataGridView و شريط أدوات للنافذة

من نافذة Data Sources انقر إشارة + بجانب الجدول Customers لتظهر لك قائمة بالحقول الخاصة به كما نلاحظ وجود نسخة من الجدول Wadaeaa كجدول فرعي ضمن Customers وذلك بسبب العلاقة التي قمنا بإنشائها بين الجدولين الآن قم بسحب الجدول Wadaeaa الموجود كجدول فرعي لـ Customers وليس الجدول الخارجي إلى سطح Form1 ليتم إنشاء DataGridView أخرى على النافذة ثم قم بتنسيق النافذة بشكل جيد وتأكد من أن الـ DataGridView الخاصة بـ Customers في الأعلى و الأخرى في الأسفل

اختر CustomersDataGridView وانقر على السهم الصغير الذي يظهر في زاويتها اليمينية العليا واختر الأمر Edit Columns واضبط الخاصية Visible للحقل ID إلى False ثم كرر العملية بالنسبة للحقل CustomerID في WadaeaaDataGridView

في شريط الأدوات في الأعلى انقر بزر الفأرة اليميني على زر الحفظ – يمتلك أيقونة قرص – واختر الأمر Enabled لتفعيله ثم انقر عليه نقرا مزدوجا لننتقل إلى محرر الكود ثم أدخل الكود التالي في حدث النقر على زر الحفظ حيث نستخدم الوظيفة WriteXml لتخزين محتويات مجموعة البيانات في ملف xml

```
Try
    MyDataSet.WriteXml("d:\TestData.xml")
Catch ex As Exception
    MsgBox(ex.Message)
End Try
```

أنشئ إجراء لمعالجة الحدث Load للنموذج وأدخل فيه الكود التالي الذي سيقوم بتحميل البيانات من ملف xml لاحظ ظهور رسالة خطأ عند تشغيل البرنامج لأول مرة وقبل حفظ البيانات حيث أن ملف البيانات لم يتم إنشاؤه بعد وهذا السبب في استخدام بلوك Try ... Catch من أجل اصطياد الخطأ وتجنب إفشال عملية بدء البرنامج وقد استخدمنا الوظيفة ReadXml لتحميل البيانات من ملف xml إلى مجموعة البيانات

```
Try
    MyDataSet.Clear()
    MyDataSet.ReadXml("d:\TestData.xml")
Catch ex As Exception
    MsgBox(ex.Message)
End Try
```

شغل البرنامج وأدخل فيه بعض البيانات في كلا الـ DataGridView وتأكد من أنك قد قمت بتعبئة جميع الحقول في كلتا شبكتي البيانات وانتبه إلى أن الحقل WadeeaPeriod هو عبارة عن عدد أشهر فترة الوديعة لذا حاول الالتزام بالقيم 1 أو 3 أو 6 أو 12 كقيمة لهذا الحقل من أجل تجربة إجرائية الاحتساب لاحقا وقم بالحفظ وأغلق البرنامج ثم أعد فتحه من جديد للتأكد من أن عملية الحفظ قد تمت بشكل صحيح لاحظ عدم ظهور رسالة الخطأ التي ظهرت عند فتح البرنامج لأول مرة بعد أن أدخلنا بيانات وقمنا بحفظها عند تشغيل البرنامج للمرة الثانية

نريد الآن إظهار قيم الاحتسابات الخاصة بكل وديعة عند المرور عليها وكذلك تاريخ استحقاق هذه الوديعة

أضف أربعة حقول نصية للنافذة ورتبها أسفل شبكتي البيانات وأعطها الأسماء التالية txtEndDate لتاريخ الاستحقاق و txtInterest لقيمة الفائدة و txtRayaa لضريبة الريع و txtIdara لضريبة الإدارة المحلية ثم انتقل لمحرر الكود وأضف الاستيراد التالي في بداية ملف الكود الخاص بـ Form1 من أجل تمكيننا من استخدام الوظائف الموجودة في مجال الأسماء Math

```
Imports ma = System.Math
```

ثم أضف الإجراء التالي كإجرائية للاحتساب وهنا أرجو أن تكون قد تابعت دروسي المتعلقة بـ Lambda Expressions لأنها الأساس في إجرائية الاحتساب

```
Private Sub DisplayWaeaaCalcs(ByVal Amount As Integer, ByVal StartDate As Date, _
       ByVal EndDate As Date, ByVal Interest As Decimal)

        Dim DaysNum = DateDiff(DateInterval.Day, StartDate, EndDate)
        Dim Rayaa As Decimal

        Dim Rayya_Calc = Function(Intrst As Decimal) ma.Ceiling(Intrst * 7.5 / 100)
        Dim Idara_Calc = Function() ma.Ceiling(Rayaa * 10 / 100)
```

```
        Dim Intrst_Calc = Function() _
                          ma.Ceiling(Amount * DaysNum * Interest / 36500)

        Dim Intr = Intrst_Calc()
        Rayaa = Rayya_Calc(Intr)
        Dim Ida = Idara_Calc()

        Me.txtEndDate.Text = EndDate.ToString("dd/MM/yyyy")
        Me.txtInterest.Text = Intr.ToString("#,###.00")
        Me.txtRayaa.Text = Rayaa.ToString("#,###.00")
        Me.txtIdara.Text = Ida.ToString("#,###.00")
    End Sub
```

حيث استخدمنا في البداية الدالة DateDiff للحصول على عدد أيام الفترة التي سنقوم بالاحتساب عنها وكنا قد مررنا قيم المبلغ و تاريخ البداية وتاريخ النهاية ونسبة الفائدة كمحددات لإجرائية الاحتساب ثم عرفنا تعبير لمدا يقوم باحتساب قيمة ضريبة الريع Rayya_Calc بناء على مبلغ الفائدة الممررة له وفي تعبير لمدا والضريبة الأخرى لم نمرر لها قيمة ولكنها استخدمت متغير محلي من أجل الاحتساب وهنا أنصحك بمراجعة قسم رفع المتغيرات في مواضيعي المتعلقة بتعابير لمدا إن لم تكن قد قرأته حتى الآن وتعبير لمدا الأخير Intrst_Calc يستخدم أيضا خاصية رفع المتغيرات ولكنه هنا يستخدم المحددات الممررة للإجراء كمتغيرات مرفوعة ثم نقوم باستخدام هذه التعابير للاحتساب ثم نظهر القيم في صناديق النصوص المناسبة

من أجل أن نقوم باحتساب القيم الموافقة لكل وديعة عند المرور عليها سنقوم بعمل إجراء معالجة للحدث CellEnter لكلا شبكتي البيانات بإجراء واحد – أدخل الكود التالي كإجراء لمعالجة الحدث CellEnter لكلا الشبكتين لاحظ ما بعد عبارة Handles في بداية تعريف جسم الإجراء وأيضا أنني لم أقم بتمرير أية محددات لإجراء معالجة الحدث حيث يمكنني فعل ذلك بما أنني متأكد من أنني لن أحتاج لاستخدامها

```
Private Sub WadaeaaDataGridView_CellEnter() _
                        Handles WadaeaaDataGridView.CellEnter, CustomersDataGridView.CellEnter

        Try
            Dim EdDat = From a In MyDataSet.Wadaeaa _
                Where a.CustomerID = Me.CustomersDataGridView.CurrentRow.Cells(0).Value _
                And a.WadeaaNumber = Me.WadaeaaDataGridView.CurrentRow.Cells(0).Value _
                Select a.WadeaaNumber, a.WadeaaAmount, a.StartDate, _
                EndDate = DateAdd(DateInterval.Month, a.WadeeaPeriod, a.StartDate), _
                a.InterestRate

            If EdDat.Count > 0 Then
                DisplayWaeaaCalcs(EdDat.First.WadeaaAmount, EdDat.First.StartDate, _
                                  EdDat.First.EndDate, EdDat.First.InterestRate)
            End If
        Catch ex As Exception
            Me.txtEndDate.Text = String.Empty
            Me.txtInterest.Text = String.Empty
            Me.txtRayaa.Text = String.Empty
            Me.txtIdara.Text = String.Empty
        End Try
    End Sub
```

في البداية قمنا بإنشاء استعلام Linq للحصول على القيم الخاصة بالوديعة التي نقف عليها حيث أن المتغير a هو كيان من الجدول Wadaeaa ثم في قسم Where ضبطنا الشرط بحيث يجلب الاستعلام فقط الودائع الخاصة بزبون معين عن طريق التأكد من أن قيمة الحقل CustomerID مساوية لقيمة ID الخاصة بالزبون من خلال قراءة قيمة الخلية المناسبة في السطر الحالي وتتمة للشرط وبنفس الطريقة قمنا بضبط الشرط كي يجلب الوديعة ذات الرقم المراد ثم يأتي قسم Select لنحدد فيه قائمة الحقول التي نريد الحصول عليها لاحظ وجود الحقل المحسوب EndDate الذي يتم حساب قيمته من الحقول المعادة من الاستعلام باستخدام الوظيفة DateAdd التي تضيف فترة زمنية معينة حسب المحددات الممررة لها إلى تاريخ ممرر لها وتعيد قيمة التاريخ الجديد وتعاد قيمته مع قائمة الحقول التي يعيدها الاستعلام وبعد الاستعلام نتأكد من أنه قد جلب نتائج فعلا بالتحقق من قيمة الخاصية Count ثم نستدعي الوظيفة DisplayWaeaaCalcs للقيام بالحسابات وإظهار النتائج

شغل البرنامج ولاحظ ظهور قيم الاحتسابات في مربعات النصوص عند التنقل في كلا شبكتي البيانات إذا كانت لديك بيانات قمت بحفظها كما طلبت منك سابقا

من أجل إظهار الودائع التي تبدأ بتاريخ معين وإظهارها أضف نموذج آخر للمشروع باسم Form2 ثم أضف DataGridView له واضبط الخاصية Dock للقيمة Fill لجعل شبكة البيانات تملأ كامل مساحة النموذج ثم نسق النموذج بحيث يكون كبيرا كفاية لعرض البيانات الناتجة عن الاستعلام ثم أضف زرا للنموذج From1 واجعل إجراء معالجة حدث النقر عليه يماثل الكود التالي

```
    Private Sub Button1_Click() Handles Button1.Click
```

```
        Dim d As Date = CDate(InputBox("Enter Date"))
        Dim Dawa = From a In MyDataSet.Wadaeaa _
                   Join B In MyDataSet.Customers On _
                   a.CustomerID Equals B.ID _
                   Where a.StartDate = d _
                   Select B.CustomerName, B.CurrentAccountNumber, _
                   a.WadeaaNumber, a.WadeaaAmount, a.WadeeaPeriod, a.StartDate, _
                   EndDate = DateAdd(DateInterval.Month, a.WadeeaPeriod, a.StartDate)

        Dim c As New Form2
        c.DataGridView1.DataSource = Dawa.ToList
        c.DataGridView1.Update()
        c.ShowDialog()
    End Sub
```

لاحظ أنني استخدمت ميزة جديدة في فيجول ستوديو تمكنني من حذف محددات إجراء معالجة حدث ما إن كنت على يقين أنني لن أحتاج لاستخدامها وفي جملة الاستعلام تلاحظ أنني استخدمت Join للربط بين الجداول عند عملية الاستعلام كي نتجنب مشكلة ظهور بيانات مكررة من أحد الجداول من أجل جميع سطور الجدول الآخر حيث استخدمنا نفس أسلوب العلاقة التي قمنا بإنشائها في البداية بين الجدولين من حيث ربط الحقل CustomerID في الجدول Wadaeaa بالحقل ID في الجدول Customer و استخدمنا في قسم Where شرط لتصفية نتائج الاستعلام بحيث نحصل على الودائع التي تبدأ بتاريخ معين ثم نستخدم عبارة Select لتحديد الحقول التي نريد إظهارها كنتائج للاستعلام

ومن أجل إظهار النتائج في Form2 قمنا بإنشاء متغير من نوع تلك النافذة ثم ضبطنا قيمة DataSource لشبكة البيانات الموجودة على ذلك النموذج إلى النتيجة المعادة من الاستعلام مستخدمين الطريقة ToList لتحويل النتائج إلى شكل يمكن إظهاره في شبكة البيانات ومن أجل الحصول على الودائع التي تنتهي بتاريخ معين يمكننا استخدام نفس الكود السابق بعد تعديل بسيط في قسم Where بحيث يصبح الكود كما يلي

```
    Private Sub Button2_Click() Handles Button2.Click
        Dim d As Date = CDate(InputBox("Enter Date"))
        Dim Dawa = From a In MyDataSet.Wadaeaa _
                   Join B In MyDataSet.Customers On _
                   a.CustomerID Equals B.ID _
                   Where DateAdd(DateInterval.Month, a.WadeeaPeriod, a.StartDate) = d _
                   Select B.CustomerName, B.CurrentAccountNumber, a.WadeaaNumber, _
                   a.WadeaaAmount, a.WadeeaPeriod, a.StartDate, _
                   EndDate = DateAdd(DateInterval.Month, a.WadeeaPeriod, a.StartDate)

        Dim c As New Form2
        c.DataGridView1.DataSource = Dawa.ToList
        c.DataGridView1.Update()
        c.ShowDialog()

    End Sub
```

لاحظ الاختلاف في قسم Where بين الإجراءين الأخيرين. هل يمكنك شرح عمل الإجراء الثاني بنفسك ؟؟؟؟

مقدمة في Linq to XML

Linq to Xml هي واجهة برمجة xml في الذاكرة تدعم Linq تمكنك من العمل مع بيانات xml المختلفة من داخل لغة برمجة الـ .net Framework وهي مشابهة لـ Document Object Model واختصارا DOM التي تضع الـ xml في الذاكرة حيث يمكنك الاستعلام من الوثيقة أو التعديل عليها ثم يمكنك حفظها أو إرسالها بعد التعديل ولكن تختلف Linq to xml عن DOM في أنها تزود نموذج غرضي Object Model أخف وأسهل عند العمل عليه وهي تستفيد من تطويرات اللغة في الـ2008

وتكمن الميزة الأهم التي تقدمها Linq to Xml هي التكامل مع Linq الذي يمكنك من كتابة استعلامات من وثيقة xml في الذاكرة للحصول على مجموعة من العناصر والصفات التي تمتد لتشمل xPath و Xquery وتقدم لك ميزات إضافية مثل اكتشاف الأخطاء في وقت الترجمة ودعم أفضل للمدقق Debugger إضافة إلى ترميز أقوى وإمكانية استخدام نتائج الاستعلامات كوسائط لبانيات XElement أو XAttribute توفر طريقة سهلة لإنشاء أشجار xml وهي تدعى Functional Construction التي تمكن المطورين بسهولة من تحويل أشجار xml بسهولة من شكل إلى آخر.

وتمكنك إمكانيات Linq في Linq to xml من كتابة استعلامات من xml فقد يكون لديك ملف xml يمثل طلب مشتريات كالتالي

PurchaseOrder.xml

```
<?xml version="1.0"?>
<PurchaseOrder PurchaseOrderNumber="99503" OrderDate="1999-10-20">
  <Address Type="Shipping">
    <Name>Ellen Adams</Name>
    <Street>123 Maple Street</Street>
    <City>Mill Valley</City>
    <State>CA</State>
    <Zip>10999</Zip>
    <Country>USA</Country>
  </Address>
  <Address Type="Billing">
    <Name>Tai Yee</Name>
    <Street>8 Oak Avenue</Street>
    <City>Old Town</City>
    <State>PA</State>
    <Zip>95819</Zip>
    <Country>USA</Country>
  </Address>
  <DeliveryNotes>Please leave packages in shed by driveway.</DeliveryNotes>
  <Items>
    <Item PartNumber="872-AA">
      <ProductName>Lawnmower</ProductName>
      <Quantity>1</Quantity>
      <USPrice>148.95</USPrice>
      <Comment>Confirm this is electric</Comment>
    </Item>
    <Item PartNumber="926-AA">
      <ProductName>Baby Monitor</ProductName>
      <Quantity>2</Quantity>
      <USPrice>39.98</USPrice>
      <ShipDate>1999-05-21</ShipDate>
    </Item>
  </Items>
</PurchaseOrder>
```

فباستخدام Linq to xml يمكنك تشغيل استعلام للحصول على القيمة المقابلة للصفة PartNumber من أجل كل عنصر في طلب المشتريات

```
Dim purchaseOrder As XElement = XElement.Load("PurchaseOrder.xml", LoadOptions.SetBaseUri
Or LoadOptions.SetLineInfo)

Dim partNos = _
    From item In purchaseOrder...<Item> _
    Select item.@PartNumber
```

وقد تريد الحصول على قائمة مرتبة باستخدام PartNumber بالعناصر التي تحمل القيمة أكثر من 100 وللحصول على هذه المعلومات يمكننا كتابة الاستعلام

```
Dim partNos = _
    From item In purchaseOrder...<Item> _
    Where (item.<Quantity>.Value * _
           item.<USPrice>.Value) > 100 _
    Order By item.<PartNumber>.Value _
    Select item
```

وباستخدام Linq to xml يمكنك عمل العديد من الأشياء كتحميل ملف من القرص أو تخزين ملف إلى القرص أو إنشاء بيانات xml من الصفر أو الاستعلام باستخدام Xpath أو حتى التعامل مع أشجار xml من حيث الإضافة والحذف والتعديل والتأكد من صحة أشجار xml باستخدام XSD أو استخدام مجموعة مما ورد هنا لتحويل أشجار xml من شكل إلى آخر

وهناك طريقتان لإنشاء أشجار xml في Visual Basic إما بتعريف xml مباشرة في الكود أو باستخدام Linq APIs لإنشاء الشجرة وكلتا الطريقتين تمكنان الكود من عكس بنية xml شجرية كاملة فالكود التالي مثلا ينشئ عنصر xml

```
Dim contact1 As XElement = _
    <contact>
      <name>Patrick Hines</name>
      <phone type="home">206-555-0144</phone>
      <phone type="work">425-555-0145</phone>
    </contact>
```

ويقدم فيجول بايزيك عدة خصائص للتنقل عبر بنية xml والتي تمكنك من الوصول إلى عناصر وصفات xml عن طريق تحديد اسم عنصر xml الابن أو يمكنك استدعاء طرائق Linq لتحديد العناصر الأبناء لعنصر xml فالكود التالي مثلا يستخدم خصائص xml للإشارة إلى الصفات والعناصر الأبناء لعنصر xml مستخدما استعلام Linq للحصول على العناصر الأبناء وإخراجهم كعنصر xml

```
' Place Imports statements at the top of your program.
Imports <xmlns:ns="http://SomeNamespace">

Module Sample1

    Sub SampleTransform()

        ' Create test by using a global XML namespace prefix.

        Dim contact = _
            <ns:contact>
                <ns:name>Patrick Hines</ns:name>
                <ns:phone ns:type="home">206-555-0144</ns:phone>
                <ns:phone ns:type="work">425-555-0145</ns:phone>
            </ns:contact>

        Dim phoneTypes = _
          <phoneTypes>
              <%= From phone In contact.<ns:phone> _
                  Select <type><%= phone.@ns:type %></type> _
              %>
          </phoneTypes>

        Console.WriteLine(phoneTypes)
    End Sub

End Module
```

ويمكنك من Visual Basic تحديد اسم مستعار Alias لمجال أسماء Xml باستخدام عبارة Imports كما في الكود التالي الذي يرينا كيف يمكننا استخدام العبارة Imports لاستيراد مجال أسماء XML

```
Imports <xmlns:ns="http://someNamespace">
```

حيث يمكنك استخدام هذا الاسم المستعار للوصول إلى خصائص xml ولتحديد محارف xml من أجل وثيقة وعناصر xml ويمكننا الحصول على غرض XNamespace من أجل أي بادئة مجال أسماء باستخدام المعامل GetXmlNamespace كما في المثال

```
' Place Imports statements at the top of your program.
Imports <xmlns:ns="http://SomeNamespace">

Module GetXmlNamespaceSample
```

```
    Sub RunSample()

        ' Create test by using a global XML namespace prefix.

        Dim contact = _
            <ns:contact>
                <ns:name>Patrick Hines</ns:name>
                <ns:phone ns:type="home">206-555-0144</ns:phone>
                <ns:phone ns:type="work">425-555-0145</ns:phone>
            </ns:contact>

        ShowName(contact.<ns:phone>(0))
    End Sub

    Sub ShowName(ByVal phone As XElement)
        Dim qualifiedName = GetXmlNamespace(ns) + "contact"
        Dim contact = phone.Ancestors(qualifiedName)(0)
        Console.WriteLine("Name: " & contact.<ns:name>.Value)
    End Sub

End Module
```

ويرينا المثال التالي كيفية إنشاء XElemnt باستخدام مجال الأسماء العام ns

```
Dim contact1 As XElement = _
    <ns:contact>
        <ns:name>Patrick Hines</ns:name>
        <ns:phone type="home">206-555-0144</ns:phone>
        <ns:phone type="work">425-555-0145</ns:phone>
    </ns:contact>

Console.WriteLine(contact1)
```

ويقوم مترجم فيجول بايزيك بترجمة محارف xml التي تحتوي الأسماء المستعارة لمجالات أسماء Xml إلى الكود المكافئ الذي يستخدم تدوين Xml المستخدم في تلك المجالات وباستخدام الصفة xmlns عند الترجمة والكود السابق يولد نفس الكود التنفيذي الذي يولده الكود التالي

```
Dim contact2 As XElement = _
    <ns1:contact xmlns:ns1="http://someNamespace">
        <ns1:name>Patrick Hines</ns1:name>
        <ns1:phone type="home">206-555-0144</ns1:phone>
        <ns1:phone type="work">425-555-0145</ns1:phone>
    </ns1:contact>

Console.WriteLine(contact2)
```

يمكن استخدام مجالات أسماء Xml العامة مع خصائص Xml كما في المثال التالي

```
Console.WriteLine("Contact name is: " & contact1.<ns:name>.Value)
```

بعض استخدامات Linq TO XML

يمكننا استخدام Linq لإنشاء وثائق xml في فيجول بايزك انظر الكود التالي الذي يقوم بإنشاء وثيقة xml تحتوي معلومات عن العمليات الجارية في النظام

```
Dim xmlProc = <MyProcesses>
                  <%= From proc In System.Diagnostics.Process.GetProcesses() _
                      Select <process id=<%= proc.Id %>>
                                 <name><%= proc.ProcessName %></name>
                                 <threads><%= proc.Threads.Count %></threads>
                             </process> %>
              </MyProcesses>

My.Computer.FileSystem.WriteAllText("d:\temp\processes.xml", xmlProc.ToString, False)
Process.Start("d:\temp\processes.xml")
```

حيث أنشأنا العقدة MyProcesses وأدخلنا فيها بداية الاستعلام فيها ثم في قسم select أنشأنا العقد الفرعية وأدخلنا بقية الاستعلام ليقوم بضبط قيم تلك العقد. كما يمكننا فيجول بايزيك من الحصول على معلومات عن العقد في وثيقة xml بسهولة حيث نستخدم الكود التالي لإظهار معلومات من الوثيقة التي قمنا بإنشائها سابقا في ListBox

```
Dim xmlprocs1 = xmlProc...<process>
For Each a In xmlprocs1
    Me.ListBox1.Items.Add(a.<name>.Value & " " & a.@id)
Next
```

أو يمكننا الحصول على نفس النتيجة السابقة باستخدام استعلامات Linq للاستعلام من وثيقة xml كما في الكود

```
Dim xmlprocs1 = From pr In xmlProc...<process> _
                Select pr.<name>.Value, pr.@id

For Each a In xmlprocs1
    Me.ListBox1.Items.Add(a.name. & " " & a.id)
Next
```

وقد نريد إنشاء وثيقة xml كنتيجة لاستعلام من وثيقة موجودة سابقا على القرص ويجب الانتباه إلى أن الاستعلام من وثائق xml حساس لحالة الأحرف

```
Dim myCusts = XDocument.Load("c:\MyCustomers.xml")

Dim ukCustomers = <ukCustomers>
                      <%= From cust In myCusts...<Customer> _
                          Where cust.<Country>.Value = "UK" _
                          Select cust %>
                  </ukCustomers>
```

أو يمكننا كتابة استعلام مباشر من وثيقة xml كما يلي

```
Dim xmlPlant = XDocument.Load(CurDir() & "\plants.xml")

Dim qa = From p In xmlPlant...<PLANT> _
         Select name = p.<COMMON>.Value _
         Order By name
```

وقد نريد كتابة استعلام من ذلك الملف لنحصل على النباتات التي تكلف أكثر من 5

```
Dim qb = From p In xmlPlant...<PLANT> _
         Where CInt(p.<PRICE>.Value) > 5 _
         Select name = p.<COMMON>.Value
```

وقد نريد إنشاء وثيقة إكسل من استعلام Linq To xml لذا افتح Excel وادخل في الثلاث خلايا الأولى من السطر الأول الكلمات التالية بالتسلسل Name و Phone و Country ثم اجعل الخط سميكا ثم أدخل تحتها سطرا من البيانات التجريبية كي نستخدمه في تحديد القسم

الذي سنضع فيه بياناتنا لاحقا ثم احفظ الملف بصيغة XML SpreadSheet ثم قم بفتح الملف الذي أنشأته للتو بواسطة Notepad وانسخ جميع محتوياته ثم عد إلى محرر الكود في بيئة التطوير واكتب = Dim Sheet ثم ألصق بعدها محتويات الحافظة فيصبح لديك شيئا شبيها بالتالي

```
Dim Sheet = <?xml version="1.0"?>
            <?mso-application progid="Excel.Sheet"?>
            <Workbook xmlns="urn:schemas-microsoft-com:office:spreadsheet"
                xmlns:o="urn:schemas-microsoft-com:office:office"
                xmlns:x="urn:schemas-microsoft-com:office:excel"
                xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet"
                xmlns:html="http://www.w3.org/TR/REC-html40">
                <DocumentProperties xmlns="urn:schemas-microsoft-com:office:office">
                    <Author>SamerSelo</Author>
                    <LastAuthor>SamerSelo</LastAuthor>
                    <Created>2008-09-19T20:31:43Z</Created>
                    <Version>12.00</Version>
                </DocumentProperties>
                <ExcelWorkbook xmlns="urn:schemas-microsoft-com:office:excel">
                    <WindowHeight>7140</WindowHeight>
                    <WindowWidth>15255</WindowWidth>
                    <WindowTopX>120</WindowTopX>
                    <WindowTopY>150</WindowTopY>
                    <ProtectStructure>False</ProtectStructure>
                    <ProtectWindows>False</ProtectWindows>
                </ExcelWorkbook>
                <Styles>
                    <Style ss:ID="Default" ss:Name="Normal">
                        <Alignment ss:Vertical="Bottom"/>
                        <Borders/>
                        <Font ss:FontName="Arial" x:CharSet="178" x:Family="Swiss" ss:Size="11"
                            ss:Color="#000000"/>
                        <Interior/>
                        <NumberFormat/>
                        <Protection/>
                    </Style>
                </Styles>
                <Worksheet ss:Name="ورقة1" ss:RightToLeft="1">
                    <Table ss:ExpandedColumnCount="3" ss:ExpandedRowCount="2" x:FullColumns="1"
                        x:FullRows="1" ss:DefaultColumnWidth="54" ss:DefaultRowHeight="14.25">
                        <Row>
                            <Cell><Data ss:Type="String">Name</Data></Cell>
                        <Cell><Data ss:Type="String">Phone</Data></Cell>
                             <Cell><Data ss:Type="String">Country</Data></Cell>
                        </Row>
                         <Row>
                         <Cell><Data ss:Type="String">Test</Data></Cell>
                              <Cell><Data ss:Type="Number">123456</Data></Cell>
                         <Cell><Data ss:Type="String">Syr</Data></Cell>
                         </Row>
                    </Table>
                    <WorksheetOptions xmlns="urn:schemas-microsoft-com:office:excel">
                        <PageSetup>
                            <Header x:Margin="0.3"/>
                            <Footer x:Margin="0.3"/>
                            <PageMargins x:Bottom="0.75" x:Left="0.7" x:Right="0.7" x:Top="0.75"/>
                        </PageSetup>
                        <Selected/>
                        <DisplayRightToLeft/>
                        <Panes>
                            <Pane>
                                <Number>3</Number>
                                <ActiveRow>1</ActiveRow>
                                <ActiveCol>2</ActiveCol>
                             </Pane>
                        </Panes>
                        <ProtectObjects>False</ProtectObjects>
                        <ProtectScenarios>False</ProtectScenarios>
                    </WorksheetOptions>
                </Worksheet>
                <Worksheet ss:Name="ورقة2" ss:RightToLeft="1">
                    <Table ss:ExpandedColumnCount="1" ss:ExpandedRowCount="1" x:FullColumns="1"
                        x:FullRows="1" ss:DefaultColumnWidth="54" ss:DefaultRowHeight="14.25">
                    </Table>
                    <WorksheetOptions xmlns="urn:schemas-microsoft-com:office:excel">
                        <PageSetup>
                            <Header x:Margin="0.3"/>
                            <Footer x:Margin="0.3"/>
                            <PageMargins x:Bottom="0.75" x:Left="0.7" x:Right="0.7" x:Top="0.75"/>
```

```
                </PageSetup>
                <DisplayRightToLeft/>
                <ProtectObjects>False</ProtectObjects>
                <ProtectScenarios>False</ProtectScenarios>
            </WorksheetOptions>
        </Worksheet>
        <Worksheet ss:Name="ورقة3" ss:RightToLeft="1">
            <Table ss:ExpandedColumnCount="1" ss:ExpandedRowCount="1" x:FullColumns="1"
                x:FullRows="1" ss:DefaultColumnWidth="54" ss:DefaultRowHeight="14.25">
            </Table>
            <WorksheetOptions xmlns="urn:schemas-microsoft-com:office:excel">
                <PageSetup>
                    <Header x:Margin="0.3"/>
                    <Footer x:Margin="0.3"/>
                    <PageMargins x:Bottom="0.75" x:Left="0.7" x:Right="0.7" x:Top="0.75"/>
                </PageSetup>
                <DisplayRightToLeft/>
                <ProtectObjects>False</ProtectObjects>
                <ProtectScenarios>False</ProtectScenarios>
            </WorksheetOptions>
        </Worksheet>
    </Workbook>
```

نلاحظ في بداية وثيقة xml التي ألصقناها للتو وجود بعض مجالات الأسماء الخاصة بـ xml في بدايتها وسنحتاج لاستيرادها في بداية كودنا لذا في قسم الاستيرادات في بداية الملف أدخل الاستيرادات التالية حتى تساعدنا في معرفة أسماء العناصر عند كتابة الاستعلام

```
Imports <xmlns="urn:schemas-microsoft-com:office:spreadsheet">
Imports <xmlns:o="urn:schemas-microsoft-com:office:office">
Imports <xmlns:x="urn:schemas-microsoft-com:office:excel">
Imports <xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet">
```

من محرر الكود ابحث عن الجزء الذي يمثل البيانات التجريبية ثم قم بقصه وفي حالة مثالي سيكون

```
<Row>
    <Cell><Data ss:Type="String">Test</Data></Cell>
    <Cell><Data ss:Type="Number">123456</Data></Cell>
    <Cell><Data ss:Type="String">Syr</Data></Cell>
</Row>
```

أدخل الآن الاستعلام التالي قبل المتغير sheet الذي عرفناه سابقا حيث سنستخدم فيه قطعة النص المقصوصة لتشكيل شكل ناتج الاستعلام

```
Dim Customers = From Customer In db.Customers _
                Order By Customer.CompanyName _
                Select <Row>
                           <Cell><Data ss:Type="String"><%= Customer.CompanyName %></Data></Cell>
                           <Cell><Data ss:Type=" String"><%= Customer.Phone %></Data></Cell>
                           <Cell><Data ss:Type="String"><%= Customer.Country %></Data></Cell>
                       </Row>
```

ثم انتقل للمكان الذي قصصنا منه قطعة xml سابقا وقم بإدخال السطر التالي في نفس المكان

```
<%= Customers %>
```

انتقل إلى الأعلى قليلا وعدل السطر

```
<Table ss:ExpandedColumnCount="3" ss:ExpandedRowCount="2" x:FullColumns="1"
```

إلى

```
<Table ss:ExpandedColumnCount="3" ss:ExpandedRowCount=<%= Customers.Count + 1 %> x:FullColumns="1"
```

وكل ما تبقى هو الحفظ وعرض الملف وهذا يتم بالكود التالي

```
Sheet.Save("d:\temp\customers11.xml")
Process.Start("d:\temp\customers11.xml")
```

تعرف على Linq to SQL و O/R Designer

سنقوم هنا بإنشاء برنامج بسيط يعرض لنا كيفية استخدام O/R Designer لإنشاء Entity Classes للتعامل المباشر مع قاعدة بيانات موجودة في SQL Server وسأعتمد حاليا على قاعدة بيانات NorthWind الشهيرة التي يمكنك الحصول عليها بسهولة من موقع مايكروسوفت وتثبيتها لديك كما يتوجب عليك إنشاء اتصال لتلك القاعدة من داخل بيئة التطوير في نافذة Server Explorer حتى يسهل علينا العمل على كل حال موضوع تركيب قاعدة البيانات وإنشاء الاتصال خارج عن مجال دورتنا وأفترض أنه لديك بعض الأساسيات التي تساعدك في التعامل مع هكذا أمور ويمكنك مراجعة مكتبة MSDN وبعض فيديوهات مايكروسوفت التعليمية إن احتجت لمساعدة في هذه الأمور

من Solution Explorer انقر بزر الفأرة اليميني على مشروعك ومن القائمة اختر Add New ومن صندوق الحوار اختر Linq To SQL Classes ثم قم بتسمية الفئة الجديدة NorthWind.dbml ثم اضغط Add فيفتح لك واجهة O/R Designer الفارغة وإن كانت مغلقة يمكنك النقر المزدوج على NorthWind.dbml من مستكشف الحل لفتحها

وسع الاتصال الخاص بقاعدة البيانات Northwind من Server Explorer واسحب الجدول Customers إلى واجهة O/R Designer ثم قم بالحفظ وبعدها انتقل إلى نافذة Data Sources اضغط Add New Data Source فيفتح لك المعالج المألوف وفي حالتنا هذه سنختار Object بما أنه المناسب لعملنا هنا ثم نضغط Next ثم وسع العقد في نافذة المعالج واختر Customer ثم اضغط Next ثم Finish

عد إلى محرر النماذج وقم بسحب الجدول Customer من واجهة DataSources إلى سطح النافذة فيتم إنشاء شريط أدوات و DataGridView من أجلك افتح السهم الصغير أعلى يمين DataGridView واضبط الخيار Dock in parent Container لجعل شبكة البيانات تملأ كافة المساحة الفارغة في النموذج وكما تلاحظ هنا أن بيئة التطوير تسهل علينا الكثير من الأمور هنا من إنشاء للتحكمات والفئات والربط بينها مما يوفر علينا الكثير من العمل

من أجل إظهار البيانات على النموذج سنحتاج لكتابة بعض الكود لذا انتقل لمحرر الكود الخاص بالنموذج وأنشئ معالج للحدث Load للنموذج وقبل بداية تعريف الحدث Load الخاص بالنموذج أدخل المتغير التالي بحيث يكون عاما على مستوى النموذج حيث أن المتغير Db هنا هو كيان من NorthWindDataContext والتي تشكل اتصالنا الفعلي مع قاعدة البيانات بما أنها نقطة الدخول الرئيسية بالنسبة لـ Linq To SQL

```
Private Db As New NorthWindDataContext
```

في الحدث Load للنموذج سنضع استعلام Linq يزودنا بالبيانات التي سيتم إظهارها

```
Dim AllCustomers = From cust In Db.Customers _
                   Order By cust.CustomerID _
                   Select cust
```

هذا استعلام Linq عادي كأي استعلام Linq قمنا باستخدامه منذ بداية الدورة حتى الآن كل ما علينا لإظهار البيانات هو ضبط قيمة الخاصية DataSource لـ CustomerBindingSourec إلى استعلامنا AllCustomers أدخل السطر التالي مباشرة بعد الاستعلام ثم شغل البرنامج وتأكد من ظهور البيانات

```
Me.CustomerBindingSource.DataSource = AllCustomers
```

من أجل حفظ التعديلات التي ربما سنقوم بها إلى قاعدة البيانات عد إلى محرر النماذج واجعل زر الحفظ الموجود على شريط الأدوات Enabled ثم انقر عليه نقرا مزدوجا لإنشاء معالج لحدث النقر عليه والانتقال لمحرر الكود وأدخل الكود التالي الذي سيقوم بحفظ البيانات من أجلنا

```
Me.Validate()
Me.CustomerBindingSource.EndEdit()

Try
    Db.SubmitChanges()
    MsgBox("Changes Saved")
Catch ex As Exception
    MsgBox(ex.Message)
End Try
```

حيث استخدمنا Me.Validate أولا لجعل جميع التحكمات على النموذج أن تتحقق من قيمها ثم استدعينا الطريقة EndEdit العائدة لـ CustomerBindingSource من أجل التأكد من أن جميع عمليات التحرير على البيانات قد تم إنهاؤها ثم استخدمنا الطريقة SubmitChanges العائدة لـ DataContext التي ستقوم بحفظ البيانات فعليا إلى قاعدة البيانات واستخدمنا بلوك Try … Catch من أجل التقاط أي خطأ ربما نصادفه أثناء عملية الحفظ ومع أننا استخدمنا الطريقة SubmitChanges هنا بدون محددات إلا أنه يمكن استخدامها بتمرير محدد وحيد من نوع التعداد ConflictMode الذي يمتلك إحدى قيميتين FailOnFirstConflict التي توقف عملية تحديث البيانات إلى قاعدة البيانات عند حصول أول تضارب وهي القيمة الافتراضية و ContinueOnConflict وهي تتابع عملية الحفظ حتى لو حدثت تضاربات وتقوم بتجميع هذه التضاربات وتعيدها بعد انتهاء عملية تحديث البيانات إلى قاعدة البيانات

يمكنك عند هذه النقطة تشغيل البرنامج والتأكد من أنك تستطيع إضافة وتعديل وحذف البيانات بدون أية مشاكل

نريد الآن إضافة بعض التصفية على البيانات في النموذج ولهذا الغرض قم بإضافة صندوق نصوص وزر أوامر إلى شريط الأدوات في أعلى النموذج وانقر نقرا مزدوجا على الزر حتى ننتقل لمحرر الكود وفي إجراء معالجة حدث النقر عليه أدخل الآن الاستعلام التالي الذي أتبعناه بكود تحديث الخاصية DataSource لـ CustomerBindingSource إلى الاستعلام الجديد لإظهار النتائج

```
Dim CustNameQuery = From cust In Db.Customers _
                    Where cust.ContactName Like _
                    Me.ToolStripTextBox1.Text & "*" _
                    Order By cust.CustomerID _
                    Select cust

Me.CustomerBindingSource.DataSource = CustNameQuery
```

ملاحظة: أنا لا أقوم بشرح الاستعلامات هنا بافتراض أنك متابع ممتاز معي منذ البداية وأصبحت متآلفا مع صيغة وطريقة كتابة هذه الاستعلامات كما يمكنك كتابة أية استعلامات تريدها هنا وبأي شكل هنا تماما كما فعلنا في الدروس السابقة

Linq To Sql Master/Detail

افتح مشروع فيجول بايزيك جديد وقم بإضافة Linq To SQL Classes إليه تماما كما فعلنا في الدرس السابق وسمها NorthWind.dbml ثم من Server Explorer وسع عقدة اتصال قاعدة البيانات NorthWind وقم باختيار الجدولين Orders و Customers وقم بسحبهما معا إلى نافذة O/R Designer الأمر الذي ينشئ فئتين من أجلنا الأولى تدعى Customer و الثانية Order ونرى أن بيئة التطوير قامت بضبط العلاقة بينهما تلقائيا كما نلاحظ أن جميع خصائص كلتا الفئتين تماثل تماما الحقول الموجودة في الجدولين الموجودين في قاعدة البيانات قم بحفظ المشروع الآن قبل المتابعة

انتقل إلى نافذة Data Sources واختر Add New Data Source واختر من الصفحة الأولى في المعالج أن النوع الذي نريد الاتصال به هو Object ثم اضغط Next للانتقال للصفحة التالية من المعالج ثم وسع العقد وقم باختيار الجدول Customer فقط بدون اختيار الجدول Order ثم اضغط Next ثم Finish وستلاحظ في نافذة Data Sources أنه قد تم إدراج الفئة Customer وتظهر الفئة Orders ككائن فرعي منها ولهذا قمنا باختيار Customer فقط في المعالج

اسحب Customer إلى سطح النافذة ليتم إنشاء DataGridView على النافذة وشريط أدوات BindingNavigator في أعلى النافذة ثم قم بسحب Orders إلى سطح النافذة لإنشاء DataGridView أخرى أسفل الأولى

انتقل إلى محرر الكود الخاص بالنموذج وقم بإنشاء إجراء لمعالجة حدث Load للنموذج وقبل تعريف الإجراء مباشرة قم بإدخال المتغير العام التالي على مستوى النموذج ليكون كيانا من NorthWindDataContext والتي تشكل اتصالنا الفعلي مع قاعدة البيانات بما أنها نقطة الدخول الرئيسية بالنسبة لـ Linq To SQL

```
Private Db As New NorthWindDataContext
```

ولجعل البيانات تظهر في كلتا الشبكتين أدخل الكود التالي في إجراء الحدث Load للنموذج وقم بتجربة البرنامج

```
Me.CustomerBindingSource.DataSource = Db.Customers
```

وعند تجربة البرنامج ستجد أن بيانات Orders قد تم جلبها وهذا تقوم به من أجلنا Linq to Sql في الخلفية بسبب العلاقة الموجودة بين الجدولين عندما أنشأنا Entity Classes في بداية العمل بواسطة O/R Designer والذي يحدث فعليا هو أنك عندما تختار سجلا من Customers يتولد تلقائيا استعلام يجلب بيانات Orders المرتبطة بهذا السجل بموجب العلاقة بين الجدولين ويظهرها في شبكة البيانات الثانية

لكي نستطيع حفظ أية تعديلات أو إضافات نقوم بها على قاعدة البيانات عد إلى محرر النماذج واجعل زر الحفظ الموجود على الشريط في أعلى النموذج Enabled وانقر عليه نقرا مزدوجا من أجل إنشاء إجراء معالجة لحدث النقر عليه والانتقال لمحرر الكود وسيكون كود الحفظ شبيها بالكود الذي استخدمناه في الدرس السابق

```
Me.Validate()
Me.OrdersBindingSource.EndEdit()
Me.CustomerBindingSource.EndEdit()

Try
    Me.Db.SubmitChanges()
    MsgBox("Changes Saved.")
Catch ex As Exception
    MsgBox(ex.Message)
End Try
```

جرب البرنامج وتأكد من أن جميع عمليات الحذف والتعديل والإضافة تعمل

انتقل إلى محرر النماذج وأضف ComboBox لشريط الأدوات في أعلى النموذج الذي سنقوم بملئه بأسماء الدول المتوفرة حتى نستطيع اختيار الزبائن الموجودين في دولة معينة انتقل الآن لمحرر الكود واستبدل كامل محتويات الحدث Load للنموذج بالكود التالي الذي سيقوم

يملئ صندوق القائمة بأسماء الدول لاحظ استخدام distinct في عبارة الاستعلام كي لا يجلب لنا نتائج مكررة عندما تكون هناك نتائج متشابهة معادة من قاعدة البيانات

```
Dim CusCountry = From co In Db.Customers _
                 Where co.Country <> String.Empty _
                 Order By co.Country _
                 Select co.Country Distinct

Me.ToolStripComboBox1.Items.Clear()

For Each co In CusCountry
    Me.ToolStripComboBox1.Items.Add(co)
Next
```

أنشئ إجراء معالجة لحدث SelectedIndexChanged لـ ToolStripComboBox1 وأدخل فيه كود الاستعلام التالي لكي يتم إظهار الزبائن الموجودة في دولة معينة عند اختيارها من صندوق القائمة المركبة

```
Dim CustQuery = From co In Db.Customers _
                Where co.Country = Me.ToolStripComboBox1.SelectedItem.ToString _
                Select co

Me.CustomerBindingSource.DataSource = CustQuery
```

وللحصول على مجموع أجور الشحن لطلبات الزبون المحدد أضف صندوق نصوص أسفل النموذج ثم أنشئ إجراء لمعالجة حدث CurrentChanged الخاص بـ CustomerBindingSource وأدخل فيه الكود التالي

```
Dim ro = CType(Me.CustomerBindingSource.Current, Customer)

Dim FriSum = Aggregate ord In Db.Orders _
             Where ord.CustomerID = ro.CustomerID _
             Into Sum(ord.Freight)

Me.TextBox1.Text = FriSum.ToString
```

حيث حصلنا أولا على معلومات السجل الحالي في CustomerBindingSource وهنا اضطررنا لاستخدام CType لتحويل ناتج الخاصية Current إلى النوع المطلوب بما أنها تعيد قيمة من النوع Object وبما أننا لم نحدد نوع المتغير ro عند التصريح عنه فستقوم بذلك نيابة عنا ميزة الاستدلال المحلي على النوع ثم استخدمنا استعلام aggregate لحساب مجموع أجور شحن الطلبات الخاصة بالزبون الحالي

أضف زرا وصندوق نصوص إلى النموذج حتى نستطيع إظهار طلبات الزبون المحدد بعد تاريخ معين ولكي لانفقد الارتباط بين شبكتي البيانات انتقل إلى إجراء معالجة الحدث CurrentChanged الخاص بـ CustomerBindingSource وأدخل فيه الكود التالي قبل الكود الموجود فيه كي نعيد ضبط القيم الصحيحة لـ OrdersBindingSource بما أننا سنغيرها لاحقا عندما سننفذ استعلامنا

```
Me.OrdersBindingSource.DataSource = CustomerBindingSource
Me.OrdersBindingSource.DataMember = "Orders"
Me.TextBox2.Text = String.Empty
```

أنشئ إجراء لمعالجة حدث النقر على الزر الذي أضفناه للنموذج وأدخل فيه الكود التالي

```
Dim ro = CType(Me.CustomerBindingSource.Current, Customer)

If IsDate(Me.TextBox2.Text) Then
    Dim SpOrd = From ord In Db.Orders _
                Where ord.CustomerID = ro.CustomerID _
                And ord.OrderDate >= CDate(Me.TextBox2.Text) _
                Select ord

    Me.OrdersBindingSource.DataSource = SpOrd
Else
    Me.OrdersBindingSource.DataSource = CustomerBindingSource
    Me.OrdersBindingSource.DataMember = "Orders"
    Me.TextBox2.Text = String.Empty
End If
```

حيث حصلنا في البداية على معلومات السجل الحالي بالنسبة لـ Customers في المتغير ro ثم استخدمنا عبارة If و الدالة CDate للتحقق من أن المستخدم قد قام بإدخال تاريخ صحيح قبل المتابعة بالاستعلام وإن لم يكن تاريخا صحيحا نعيد ضبط قيم OrdersBindingSource إلى القيم الأصلية ونقوم بتفريغ النص الموجود في صندوق النصوص وإن كان قد ادخل تاريخا صحيحا ننشئ استعلام يقوم بجلب الطلبات التي تاريخها بعد التاريخ الموجود في صندوق النصوص والخاصة بالزبون الحالي كما هو ظاهر في قسم Where في الاستعلام ثم نقوم بضبط الخاصية DataSource لـ OrdersBindingSource إلى استعلامنا كي يتم عرض البيانات المعادة من الاستعلام في شبكة البيانات الثانية

مثال سريع عن كيفية إنشاء فئات Linq To SQL يدويا

رأينا في الدروس السابقة كيف أن الـ O/R Designer يقوم بإنشاء فئات Linq To SQL أو كما تدعى أيضا Entity Classes بسهولة من أجلنا ومع ذلك يمكننا القيام بذلك يدويا وسأقوم في هذا الدرس باستعراض سريع لكيفية عمل ذلك من أجل العلم بالشئ

أنشئ مشروعا جديدا من النوع Console Application وسمه LinqConsoleApp

- من قائمة Project اختر Add Reference ومن صفحة net. اختر System.Data.Linq ثم اضغط Ok
- في بداية الملف في أعلى محرر الكود أضف الاستيرادات التالية

```
Imports System.Data.Linq
Imports System.Data.Linq.Mapping
```

سنضيف الآن Entity Class والذي عن عبارة عن فئة منظمة وفق جدول قاعدة بيانات ولإنشاء هذه الفئة نضيف الصفة Table قبل تعريف الفئة والت تمتلك الخاصية Name التي تحدد اسم الجدول في قاعدة البيانات ثم سيكون علينا إضافة الخصائص المناسبة لتمثل أعمدة الجدول ويتم تحديد الربط مع الأعمدة في قاعدة البيانات باستخدام الصفة Column والتي تمتلك عدة خصائص لتعريف العمود مثل IsPrimaryKey التي تمتلك قيمة منطقية تحدد فيما إذا كان العمود مفتاح أساسي أم لا والخاصية Storage التي تحدد اسم الحقل الخاص الذي سيخزن قيمة الخاصية – أضف الفئة التالية بعد تعريف الـ Module وقبل Sub Main والتي ستمثل الجدول Customers في قاعدة البيانات طبعا عرفت بعض الخصائص من أجل بعض الحقول الموجودة في الجدول هنا وليس جميع الحقول وذلك من اجل الاختصار هنا وإن قررت استخدام هذه الطريقة عمليا عليك تعريف الفئة بحيث تكون مطابقة تماما للجدول الذي تمثله

```
<Table(Name:="Customers")> _
  Public Class Customer

      Private _CustomerID As String
      <Column(IsPrimaryKey:=True, Storage:="_CustomerID")> _
      Public Property CustomerID() As String
          Get
              Return Me._CustomerID
          End Get
          Set(ByVal value As String)
              Me._CustomerID = value
          End Set
      End Property

      Private _City As String
      <Column(Storage:="_City")> _
      Public Property City() As String
          Get
              Return Me._City
          End Get
          Set(ByVal value As String)
              Me._City = value
          End Set
      End Property

     Private _ContactName As String
      <Column(Storage:="_ContactName")> _
      Public Property ContactName() As String
          Get
              Return Me._ContactName
          End Get
          Set(ByVal value As String)
              Me._ContactName = value
          End Set
      End Property

  End Class
```

سنحتاج إلى إنشاء اتصال مع قاعدة البيانات NorthWind التي سنستخدمها لذا سنحتاج إلى تعريف غرض DataContext والذي يعتبر القناة الرئيسية لقراءة البيانات وتخزينها في قاعدة البيانات – أدخل سطر الكود التالي في الإجراء Sub Main وذلك باعتبار أن قاعدة البيانات موجودة في الملف NORTHWND.MDF الموجود في المجلد D:\TEMP لدي عند تجربة هذا المثال

```
Dim db As New DataContext("D:\TEMP\NORTHWND.MDF")
```

ثم سنقوم بإنشاء كائن Table(of Customer) كي نستخدمه في الاستعلام من الجدول Customers أدخل الكود التالي مباشرة بعد الكود السابق

```
Dim Customers As Table(Of Customer) = _
          db.GetTable(Of Customer)()
```

أضف السطر التالي من الكود الذي سيطبع على نافذة الكونسول أوامر Sql التي سيتم تنفيذها على قاعدة البيانات

```
db.Log = Console.Out
```

سنكتب الآن استعلام Linq ليستعلم أي من الزبائن موجودين في لندن – أدخل كود الاستعلام التالي بعد الكود السابق مباشرة

```
Dim custQuery = From cust In Customers _
                Where cust.City = "London" _
                Select cust
```

وللحصول على نتائج الاستعلام سنستخدم حلقة For Each للدوران عبر نتائج الاستعلام كما رأينا في الدروس السابقة ثم طباعتها على نافذة الكونسول

```
Console.WriteLine("Number Of Records: " & custQuery.Count.ToString)

For Each CustObj In custQuery
    Console.WriteLine(CustObj.CustomerID.ToString & ", " & _
                      CustObj.City & ", " & CustObj.ContactName)
Next

Console.ReadLine()
```

وهذا هو الكود الكامل للمشروع

```
Imports System.Data.Linq
Imports System.Data.Linq.Mapping

Module Module1

    <Table(Name:="Customers")> _
    Public Class Customer

        Private _CustomerID As String
        <Column(IsPrimaryKey:=True, Storage:="_CustomerID")> _
        Public Property CustomerID() As String
            Get
                Return Me._CustomerID
            End Get
            Set(ByVal value As String)
                Me._CustomerID = value
            End Set
        End Property

        Private _City As String
        <Column(Storage:="_City")> _
        Public Property City() As String
```

```
            Get
                Return Me._City
            End Get
            Set(ByVal value As String)
                Me._City = value
            End Set
        End Property

        Private _ContactName As String
        <Column(Storage:="_ContactName")> _
        Public Property ContactName() As String
            Get
                Return Me._ContactName
            End Get
            Set(ByVal value As String)
                Me._ContactName = value
            End Set
        End Property

    End Class

    Sub Main()
        Dim db As New DataContext("D:\TEMP\NORTHWND.MDF")

        Dim Customers As Table(Of Customer) = _
                db.GetTable(Of Customer)()

        db.Log = Console.Out

        Dim custQuery = From cust In Customers _
                        Where cust.City = "London" _
                        Select cust

        Console.WriteLine("Number Of Records: " & custQuery.Count.ToString)

        For Each CustObj In custQuery
            Console.WriteLine(CustObj.CustomerID.ToString & ", " & _
                              CustObj.City & ", " & CustObj.ContactName)
        Next

        Console.ReadLine()

    End Sub

End Module
```

أمثلة على استعلامات Linq

في هذا الدرس سأورد بعض الأمثلة على استعلامات مختلفة كي نتعرف أكثر على أسلوب كتابة هذه الاستعلامات ولن أتطرق إلى طرق استخدام هذه الاستعلامات بما أننا رأينا كيف يمكن عمل ذلك من خلال الأمثلة الواردة في الدروس السابقة إما بالدوران باستخدام حلقة ... For Each أو بالإسناد المباشر لمتغير أو تحكم أو الإظهار في DataGridView

مثال بسيط على استخدام الدالات التجميعية في الاستعلام للحصول على القيمة العظمى لحقل يحتوي على مجموعة قيم

```
Dim Wod = Aggregate Wdt In ads.WorkingDate _
          Into Mdt = Max(Wdt.WorkingDate)
```

مثال آخر على استخدام الدالات التجميعية للحصول على مجموع ناتج عملية طرح حقلين

```
Dim TempBal = Aggregate AccBa In AccMov _
              Into Bal = Sum(AccBa.DebitAmmount - AccBa.CreditAmmount)
```

مثال بسيط آخر يستخدم Like في قسم Where في الاستعلام للحصول على مجموعة نتائج محددة

```
Dim AccMov = From AccBal In Accds.AccountMovements _
             Where AccBal.AccountNumber Like (AccNum & "*") _
             Select AccBal
```

مثال يستخدم الدالة ToLower في قسم Where للحصول على نتائج الاستعلام بحيث لا يتأثر شرط التصفية بحالة الأحرف

```
Dim Qts = From cu In Db.Customers _
          Where cu.Country.ToString.ToLower Like "po*".ToLower _
          Select cu
```

مثال بسيط آخر يستخدم Order By لترتيب النتائج تصاعديا

```
Dim ParentList = From ParLst In AccountsDataSet.Accounts _
                 Where ParLst.IsChildAccount = False _
                 Order By ParLst.AccountNumber _
                 Select ParLst
```

ومن أجل الترتيب تنازليا يصبح الاستعلام كما يلي وذلك بإضافة Descending بعد حقل الفرز في قسم Order By

```
Dim ParentList = From ParLst In AccountsDataSet.Accounts _
                 Where ParLst.IsChildAccount = False _
                 Order By ParLst.AccountNumber Descending _
                 Select ParLst
```

هذا المثال يقوم بإعادة تاريخ محتسب من تاريخ وفترة زمنية مخزنين في قاعدة بيانات مع استخدام Join لربط جدولين للحصول على القيم منهما

```
Dim Edat = From de In DsDesposits.Deposits _
           Join Dp In DsDesposits.InterestRates _
           On de.InterestID Equals Dp.InterestID _
           Select EndDate = DateAdd(DateInterval.Month, _
                                    intrst.First, DepDet.First.StartDate)
```

مثال آخر عن استعلام يستخدم Join لربط جدولين من أجل الاستعلام منهما ثم اختيار حقول محددة كنتيجة للاستعلام ثم استخدام الاستعلام داخل استعلام آخر من أجل الحصول على نتيجة إضافية ثم استخدام Join للاستعلام من نتيجة ربط استعلام وجدول للحصول على النتائج المرغوبة

```
Dim MovNam = From Acc In SampleDatabaseDataSet.ACCOUNTS _
             Join Mov In SampleDatabaseDataSet.ACCOUNTS_MOVEMENTS _
             On Acc.ACC_NUMBER Equals Mov.ACC_NUMBER _
             Select Mov.MOVMENT_DATE, Mov.ACC_NUMBER, Acc.ACC_NAME, _
             Mov.DEBIT_AMOUNT, Mov.CREDIT_AMOUNT

Dim AccBal = From AcBa In MovNam _
             Group By acc_number = AcBa.ACC_NUMBER _
             Into Balance = Sum(AcBa.DEBIT_AMOUNT - AcBa.CREDIT_AMOUNT)


Dim AccBalName = From ab In AccBal Join ac In SampleDatabaseDataSet.ACCOUNTS _
                 On ab.acc_number Equals ac.ACC_NUMBER _
                 Select ab.acc_number, ac.ACC_NAME, ab.Balance
```

هذا المثال يستخدم Join لإنشاء استعلام يعطينا النتائج نتيجة الاستعلام من أربعة جداول مختلفة

```
Dim ViewData = From Cu In DsDesposits.Customers _
               Join Dp In DsDesposits.Deposits On Dp.CustomerID Equals Cu.CustomerID _
               Join DepPer In DsDesposits.InterestRates On DepPer.InterestID Equals Dp.InterestID _
               Join CaDp In DsDesposits.CalculatedDepsits On CaDp.DepositID Equals Dp.DepositID _
               Where CaDp.CalculationReason = 2 _
               And CaDp.CalculationDate >= FromDate _
               And CaDp.CalculationDate <= ToDate _
               Select Cu.Name, Dp.DepositNumber, Dp.DepositAmount, Dp.StartDate, _
               CaDp.CalculationDate, DepPer.DepositPreiod
```

هذا المثال فيه نقطتين الأولى بخصوص in المستخدمة في استعلامات sql للحصول على نتيجة من قائمة قيم فهي غير موجودة هنا لذا نقوم بوضع القيم في مصفوفة ثم نستخدم Contains في الاستعلام للحصول على نفس النتيجة والثانية هي أننا نستطيع استخدام نتيجة استعلام كدخل لاستعلام آخر

```
Dim Vlu() As Byte = {0, 3, 4, Nothing, 6}
Dim DepDet = From De In DsDesposits.Deposits _
             Where De.DepositID = Dr.DepositID _
             And Vlu.Contains(De.DepositStatus)

Dim Intrst = From Irs In DsDesposits.InterestRates _
             Where Irs.InterestID = DepDet.First.InterestID _
             Select Irs.DepositPreiod, Irs.InterestRate
```

مثال آخر يستخدم Join وطريقة المصفوفة معا

```
Dim Vlu() As Byte = {0, Nothing, 3, 4, 6}
mDep = From d In DsDesposits.Deposits _
       Join i In DsDesposits.InterestRates On d.InterestID Equals i.InterestID _
       Where DateAdd(DateInterval.Month, i.DepositPreiod, d.StartDate) <= Now.Date _
       And Vlu.Contains(d.DepositStatus) _
       Select d.DepositID
```

هذا المثال يستخدم استعلامين متداخلين ففي قسم select في نهاية الاستعلام الأول استخدمنا الطريقة Except لاستثناء النتائج المعادة من الاستعلام الثاني الممرر كمحدد للطريقة Except

```
Deps = (From d In DsDesposits.Deposits _
        Join i In DsDesposits.InterestRates _
        On i.InterestID Equals d.InterestID _
        Where (d.StartDate <= New Date(Ryear, 12, 31) _
        And vlu.Contains(d.DepositStatus))
        And DateAdd(DateInterval.Month, i.DepositPreiod, d.StartDate) >= New Date(Ryear, 12, 31) _
        Select d.DepositID).Except( _
        From ca In DsDesposits.CalculatedDepsits _
```

```
Where ca.EndDate = New Date(Ryear, 12, 31) _
Select ca.DepositID)
```

الاستعلامات المترجمة Compiled Queries

عندما يكون لدينا تطبيق يستخدم استعلامات متشابهة العديد من المرات يمكنك زيادة الأداء عبر ترجمة Compile الاستعلام مرة واحدة ثم تنفيذه العديد من المرات لاحقا عبر تمرير محددات مختلفة في كل مرة فقد يكون لديك تطبيق يقوم بالاستعلام عن جميع الزبائن الموجودين في مدينة محددة بحيث يتم تمرير اسم المدينة في وقت التنفيذ من قبل المستخدم وهنا تدعم Linq to Sql استخدام الاستعلامات المترجمة لهذا الغرض

حيث توفر لنا الفريموورك الفئة CompiledQuery التي تزودنا بإمكانية الترجمة والتخزين المؤقت للاستعلامات من اجل الاستخدام وهذه الفئة متواجدة في مجال الأسماء System.Data.Linq وهي تمتلك خاصية وحيدة هي Expression التي تعيد الاستعلام كتعبير لمدا Lambda Expression وهي تمتلك بعض الطرق ولكن الطريقة الأهم والتي يتم استخدامها هنا هي الطريقة Compile التي تمتلك عدة أشكال محملة متشابهة

```
Public Shared Function Compile(Of TArg0 As DataContext, TArg1, TResult) ( _
         query As Expression(Of Func(Of TArg0, TArg1, TResult)) _
         ) As Func(Of TArg0, TArg1, TResult)


Public Shared Function Compile(Of TArg0 As DataContext, TArg1, TArg2, TResult) ( _
         query As Expression(Of Func(Of TArg0, TArg1, TArg2, TResult)) _
         ) As Func(Of TArg0, TArg1, TArg2, TResult)


Public Shared Function Compile(Of TArg0 As DataContext, TArg1, TArg2, TArg3, TResult) ( _
         query As Expression(Of Func(Of TArg0, TArg1, TArg2, TArg3, TResult)) _
         ) As Func(Of TArg0, TArg1, TArg2, TArg3, TResult)


Public Shared Function Compile(Of TArg0 As DataContext, TResult) ( _
         query As Expression(Of Func(Of TArg0, TResult)) _
         ) As Func(Of TArg0, TResult)
```

حيث تمثل المحددات TArg نوع المحدد الممرر للمفوض Delegate عندما يتم تنفيذ المفوض المعاد من الطريقة Compile و المحدد TResult هو من النوع IEnumerable(T) المعاد عند تنفيذ المفوض المعاد من الطريقة Compile

في العديد من الحالات ربما ترغب في إعادة استخدام الاستعلامات خارج مجال مسار التنفيذ الحالي ففي هذه الحالات تكون عملية تخزين الاستعلامات المترجمة في متغيرات ساكنة Static Variables فكرة فعالة ففي الكود التالي توجد لدينا فئة Queries مصممة من أجل تخزين الاستعلامات المترجمة تستخدم للاستعلام من قاعدة بيانات NorthWind بطريقة Linq to Sql

```
Class Queries

    Public Shared CustomersByCity As _
        Func(Of NorthWindDataContext, String, IQueryable(Of Customer)) = _
            CompiledQuery.Compile(Function(db As NorthWindDataContext, _
            city As String) _
        From c In db.Customers Where c.City = city Select c)

    Public Shared CustomersById As _
        Func(Of NorthWindDataContext, String, IQueryable(Of Customer)) = _
            CompiledQuery.Compile(Function(db As NorthWindDataContext, _
            id As String) _
        db.Customers.Where(Function(c) c.CustomerID = id))

End Class
```

حيث يمكننا تعريف وظيفة تستخدم الاستعلام المترجم كما يلي

```
Public Function GetCustomersByCity(ByVal city As String) As _
    IEnumerable(Of Customer)
```

```
        Return Queries.CustomersByCity(db, city)
End Function
```

ثم استخدام هذه الوظيفة لعرض نتائج الاستعلام في DataGridView كما يلي

```
Me.DataGridView1.DataSource = GetCustomersByCity("London").ToList
```

أو حتى الاستخدام المباشر للاستعلام المترجم بحيث نظهر نتيجة الاستعلام مباشرة في DataGridView

```
Me.DataGridView1.DataSource = Queries.CustomersByCity(db, "London").ToList
```